INTERNET BASED DATA COLLECTION AND MONITORING FOR WIRELESS SENSOR NETWORKS

By

Venkatramana Reddy Revuri

RECOMMENDED:

Dr. Dejan Raskovic, Advisory Committee Chair

Dr. Charles E. Mayer

Dr. Seta Bogosyan

Chair, Department of Electrical and Computer Engineering

APPROVED:

Dean, College of Engineering and Mines

Dean of the Graduate School

March 31, 2007

Date

INTERNET BASED DATA COLLECTION AND MONITORING FOR WIRELESS SENSOR NETWORKS

A

THESIS

Presented to the Faculty

of the University of Alaska Fairbanks

in Partial Fulfillment of the Requirements

for the Degree of

MASTER OF SCIENCE

By

Venkatramana Reddy Revuri, B.Tech.

Fairbanks, Alaska

May 2007

ABSTRACT

The omnipresence of the Internet and the advances in integrated circuit technologies has expanded the potential modes of communication and data collection. Adding Internet capabilities to any electronic device greatly extends the device's user interface, allowing the user to remotely configure and monitor the device over the network through the embedded web server. The embedded web server is expected to establish two-way communication and serve dynamic web pages using very limited resources. We adapted an existing embedded web server to allow remote control and monitoring of wireless sensor networks (WSN). This required establishing an interface to the WSN and developing firmware and user programs to communicate with the remote client. An interactive and flexible web-based user management interface is developed to allow the two-way interaction between the remote user and the wireless sensor network. The embedded server generates email alerts to the administrator about critical issues in the WSN, provides secure access to the WSN control modules, etc. Two embedded web servers are developed using different hardware platforms. The first solution is a low cost, energy efficient solution with somewhat limited functionality. The other uses a more powerful microcontroller-based platform and implements a fully-functional, dynamic web server with multiple web pages.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF APPENDICES

1 Introduction

With the advent of the World Wide Web and the number of web users increasing rapidly, the underlying technologies can be used for much more than just web browsing. The widespread acceptance of online communication has replaced the conventional data collection and monitoring methods, which required physical presence at the research site, with remote access using the Internet. The current web technology makes it possible to remotely control and monitor applications easily and inexpensively over large distances. The web-enabled device can be monitored and remotely controlled using a dedicated firmware device. This dedicated firmware device acts as a web server and it can receive requests and commands from a remote Hypertext Transfer Protocol (HTTP) web client or a web browser. The device then sends the user management interface of the target device, in the form of a Hypertext Markup Language (HTML) page, through which the user can monitor the network parameters.

The firmware device can be integrated into a wide variety of products: for office use, it can be embedded into copiers, printers, communication and telephone devices; for home appliances, it can be embedded into televisions, video and audio players, air conditioners, microwave ovens, security systems and surveillance cameras; in laboratories, it can be used in oscilloscopes, spectrum analyzers, etc. Figure 1-1 shows the embedded web server integrated into everyday devices.

In the year 1996, Hewlett Packard filed a patent "Embedding web access mechanism in an appliance for user interface functions including a web server and web browser." The patent was issued in the year 1999. The abstract reads as follows:

> "Web access functionality is embedded in a device to enable low cost, widely accessible and enhanced user interface functions for the device. A web server in the device provides access to the user interface functions of the device through a device web page. A network interface in the device enables access to the web page by a web browser such that a user of the web browser accesses the user interface functions for the device through the web page." [1]

The patent shows the trend towards the use of remote monitoring and controlling techniques for a device. The devices are now able to easily integrate the functionality of remote access. By adding the functionality of standardized remote access, the manufacturer gains the competitive advantage.

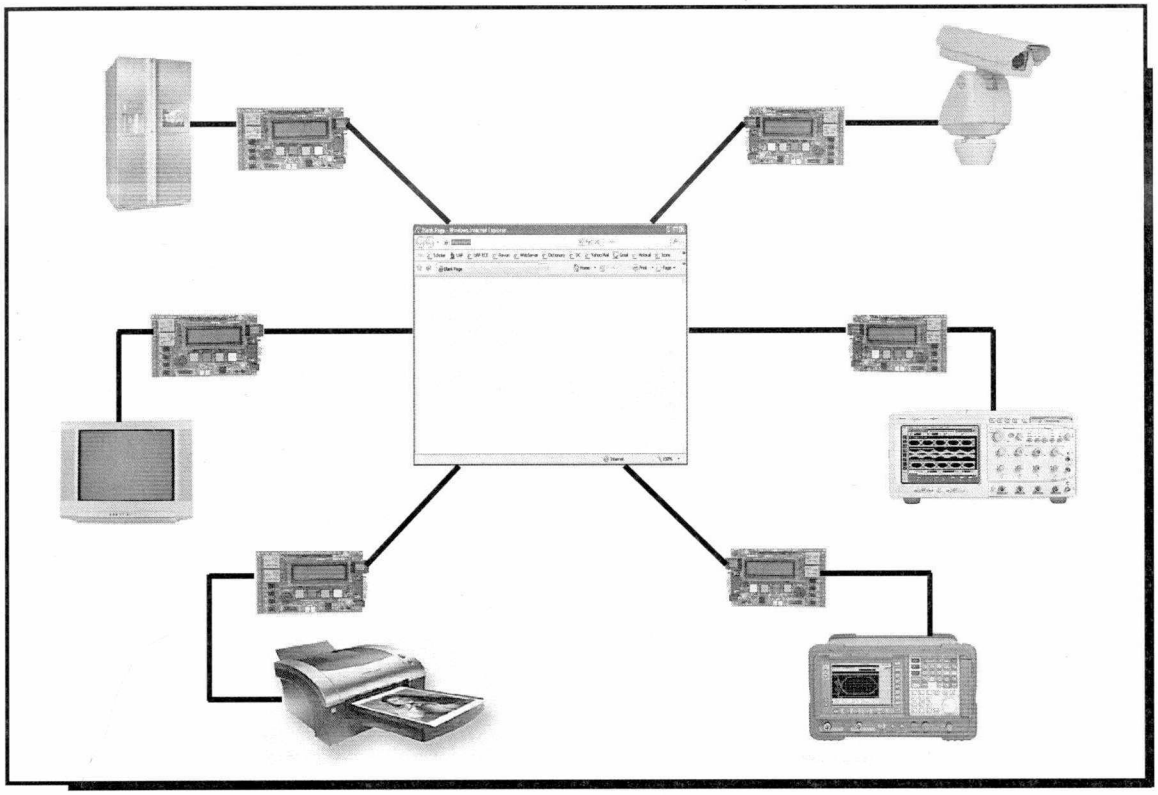


Figure 1-1: Examples of everyday devices with embedded web servers that can be controlled through any standard web browser

The majority of the support is provided by the standardized client application: the web browsers. The web browser is a software application that provides access to information on the web and offers a standardized representation model to present the information using images, highly formatted text and graphical user interface. A browser at a minimum consists of a HTML interpreter and HTTP client to request and retrieve web pages from the web server.

The Internet-based user management interface provides the user with the ability to configure, monitor and control the network device over the Internet. The user management interface is made up of basic HTML code, images and a graphical user

interface to interact with the network device over the Internet. The device home page provides the central navigation point to the device controls, and the device responds to the web page buttons, hot links, and many familiar browser controls.

In the field of wireless communication, the availability of low power, cheap and small embedded processors, radios, and sensors often integrated on a single chip, is leading to the use of wireless communications and computing to interact with the physical world in applications such as security, surveillance etc. The resulting system is called a wireless sensor network. Wireless sensors are very widely used. It is believed that someday millions of embedded chips and sensing devices will be integrated into objects and locations that are part of our daily lives.

Integrating the above breakthrough inventions in the fields of embedded systems and wireless communications, in this thesis an Internet-based user management interface is developed for data collection and control[1] of a wireless sensor network. The major difficulty in implementing the embedded web server lies in the fact that the resources available are much more limited than those typically used for the regular web server. The embedded web server is expected to establish two-way communication and to be capable of serving dynamic web pages. In this thesis an existing simple, one-way microcontroller-based embedded web server is adapted and further enhanced with the capabilities of monitoring and remotely accessing the data collected by wireless sensor nodes. This process requires developing a set of firmware and user programs to allow data collection and control network operations.

1.1 Project Overview

The initial implementation of a web server was based on the MSP430, an ultra low power microcontroller from Texas Instruments. The existing one-way communication version was extended to allow a two-way communication between the client and the web server, as well as handling multiple web page requests and responses.

---

[1] The term 'control' in this thesis refers to the control of operational parameters of the wireless sensor network such as the radio transmission power, communication channel, types of data being collected, etc.

The research proceeded further, taking up a new, faster microcontroller from Rabbit Semiconductor with increased memory capacity for storing the web pages and the data collected from the network. The capabilities of the web server were extended by adding the functionality of data collection, monitoring and network control for the wireless sensor network. The user can monitor the wireless sensor network either by connecting the server directly to the hub (central node) or connecting it to one of the wireless sensor nodes. The communication between the server and the wireless sensor network was implemented by connecting the server to one of the wireless sensor nodes. The advantage in doing so is to make the web server mobile and a wireless interface for wireless Internet connection will be a good add-on feature. Establishing a communication between the web server and one of the wireless sensor nodes required developing firmware and a communication protocol. The data packet is collected from the sensor node and stored on the serial flash memory of the web server. Figure 1-2 shows the overall setup of the project.

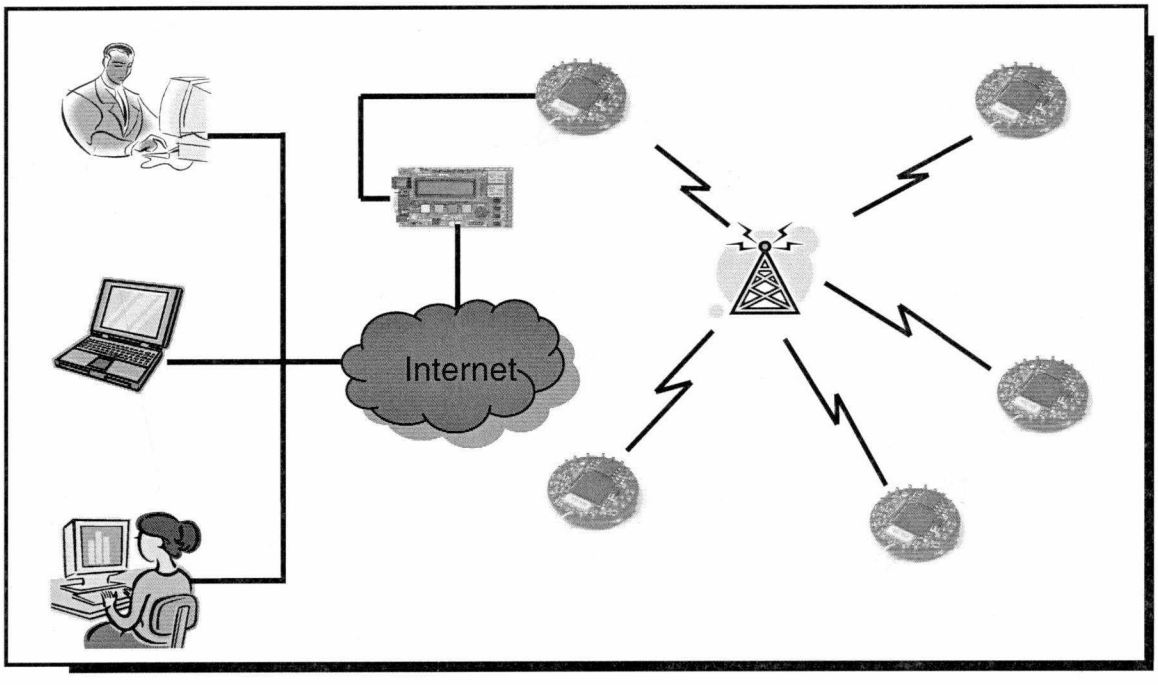


Figure 1-2: The generalized picture of an embedded web server at work

The web server is connected to one of the wireless sensor nodes of the network through which it controls, monitors, and collects the network managed data. The firmware is developed for the web server to work in three different modes of operation, which include: *Monitoring mode*, where the web server is used for monitoring the network parameters; *Log mode*, where the data is collected from the network and stored on the serial flash, and *Snapshot mode*, where the client can access specific network monitored parameter. Further, programs are written to enable transfer of serial flash data to the client machine, for network data analysis.

The web server provides administrative privileges for an administrator to control the server's mode of operation and monitor the wireless sensor network. The administrator or a registered user can set the mode of operation of the web server and can request the network parameters from the sensor network. The client sends the request to the web server for the network data or the parameters and the server processes the request and replies with the requested data embedded in the HTML page.

The organization of the rest of the thesis is as follows:

Chapter 2 covers some basics regarding the operation principle(s) of a web server, compares general web servers and embedded web servers, and discusses the embedded web server advantages and functional and non-functional requirements.

Chapter 3 discusses various research projects and commercially available embedded web servers. It also includes the descriptions of a few embedded web server applications.

Chapter 4 presents the Internet connectivity of MSP430. The chapter describes the hardware and software implementations of the MSP430-based embedded web server for wireless sensor network.

Chapter 5 discusses the Internet connectivity of RCM3750 and its advantages over the MSP430-based embedded web server. The chapter presents the Rabbit embedded web server hardware, TCP/IP stack implementation, and the web-based user management interface.

Chapter 6 presents the conclusions and future work.

2 Embedded Web Server

2.1 Basic Ethernet Protocols

Every computer network has common requirements: a physical interface to connect the computer to the network and some governing rules to enable the exchange of data between computers over the network. The physical component should include a well-defined interface to ensure that the output of the transmitting computer is compatible with the input of the receiving computer. The Ethernet standard defines this interface in the case of Ethernet networks.

The computers in the network must agree on many aspects to enable a reliable data transfer through all networks. Some of the aspects of sharing the network include rules which decide when a computer can transmit data on a shared network, addressing schemes for a frame to reach its intended destination, and a defined frame format for information exchange. The software complexity of network protocols is greatly simplified by dividing the whole process into smaller modules, or components, which work together with hardware devices to handle the job of network communication and provide a powerful communication system. Each module is a subset of the complete network protocol and performs a single task or a small set of related tasks. The module provides its functions and services to one or more other modules, but the operational details of the modules are kept hidden from each other.

The modules used in the networking are like a pile of building blocks stacked in layers one above another; hence the group of related protocols is often called a protocol stack. Each layer in the stack provides the services to its upper layer with the help of lower layer protocol. The operational details of the lower layer are hidden from the higher level protocols, making the modules independent of one another. This modular design makes the maintenance of network protocols easy, allowing modifications in any layer without affecting the other modules. In the early networking years, International Organization for Standardization (ISO) developed a seven-layer model which led to the modified five-layer Transmission Control Protocol/Internet Protocol (TCP/IP) reference

model to meet the current needs of protocol designers. Table 2-1 describes the layers of ISO reference model and its functions and Table 2-2 presents the TCP/IP five-layer Internet reference model, its functions and examples.

Table 2-1: ISO/OSI seven-layer Internet reference model

	Name of the Layer	**Description**
Layer 7	Application layer	Defines protocols for applications that use the network
Layer 6	Presentation layer	Defines protocols for presenting the data
Layer 5	Session layer	Defines protocols that manage communication sessions between applications
Layer 4	Transport layer	Defines protocols that provide end-to-end error detection and correction for reliable data transfer
Layer 3	Network layer	Defines addressing assignments to manage network connections
Layer 2	Data link layer	Defines frame format organization for reliable data delivery across the physical media
Layer 1	Physical layer	Defines network hardware interfaces

The application layer provides the data for transfer. The data packet is passed down the stack until it reaches the network layer, where it is put on the network for transmission. The intermediate layers are defined to prepare the message for transmission by adding headers and/or trailers to help ensure that the transmitted data gets to its destination. Each intermediate layer in the stack adds the control information either at the end or at the start of the transmitting message; hence the control information is called a header or trailer of the layer.

Table 2-2: TCP/IP five-layer Internet reference model

	Name of the Layer	**Description**	**Examples**
Layer 5	Application layer	Defines protocols for applications that use the network	FTP, SMTP, HTTP, DHCP, DNS, PING
Layer 4	Transport layer	Defines protocols that provide end-to-end error detection and correction for reliable data transfer	TCP, UDP, ICMP
Layer 3	Internet layer	Defines packet format and routing between network nodes	ARP, IP
Layer 2	Network layer	Defines frame organization and hosts specific transmission of datagram	Ethernet
Layer 1	Physical layer	Defines network hardware interfaces	

Each layer treats the whole message from the layer above it as a data message, and adds the control information; this wrapped message is then forwarded to the layer below it. By the time the message reaches the end of the stack, it will be enveloped in multiple wrappers, one for each layer of the protocol stack, and is then passed over to the network. This process of wrapping the message by header and/or trailer by each protocol layer of the stack is called encapsulation. Figure 2-1 shows the encapsulation of the data packet on their way from the application layer to the network layer.

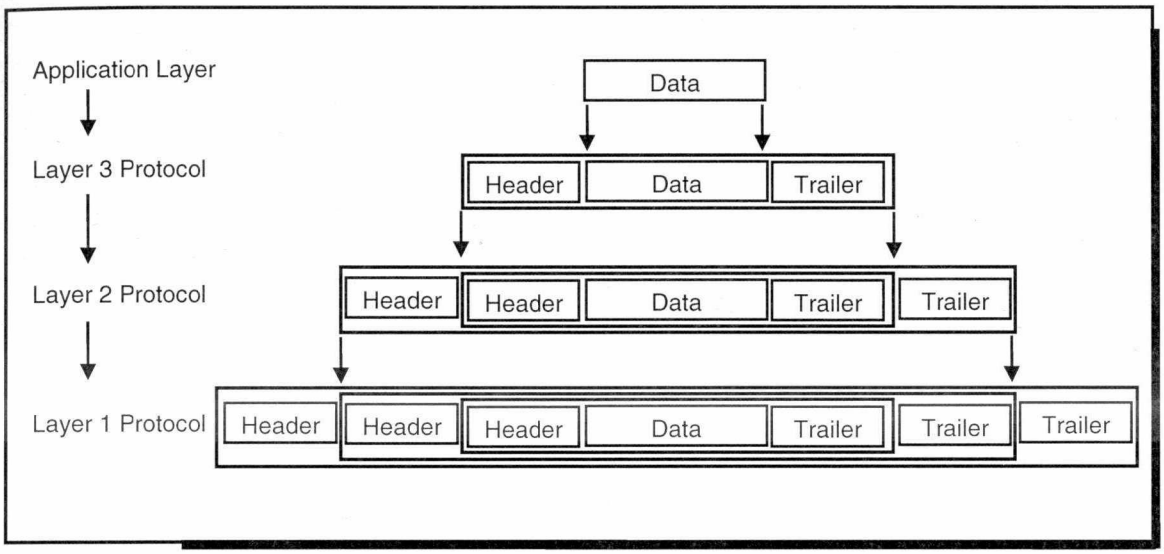


Figure 2-1: Encapsulation of data through different layers of the protocol stack

At the receiving end, the data packet is sent up the stack from the network layer to the application layer, where it is received by the application. Each intermediate layer strips off its header and/or trailer and passes the data to the overlying layer. The opposite of encapsulation, which is the process of removing the header and/trailer at each layer, is called decapsulation. The message at each layer of the sending protocol stack is identical to the corresponding layer at the receiving side. Figure 2-2 shows the virtual connection between the application layers of the sender and receiver and shows the identical message at each corresponding layer.

2.2 Transmission Control Protocol/Internet Protocol

TCP/IP is a set of rules that define the network communication process and which specifically defines the addressing procedure between two computers, the frame format, and the control information needed by the receiving computer to interpret the message correctly. The TCP/IP standard represents the governing rules which define the communication on the network. The TCP/IP implementation is the software module that functions according to the standard and enables the computers to communicate on the network. TCP/IP ensures the compatibility of all TCP/IP implementations regardless of versions or vendors.

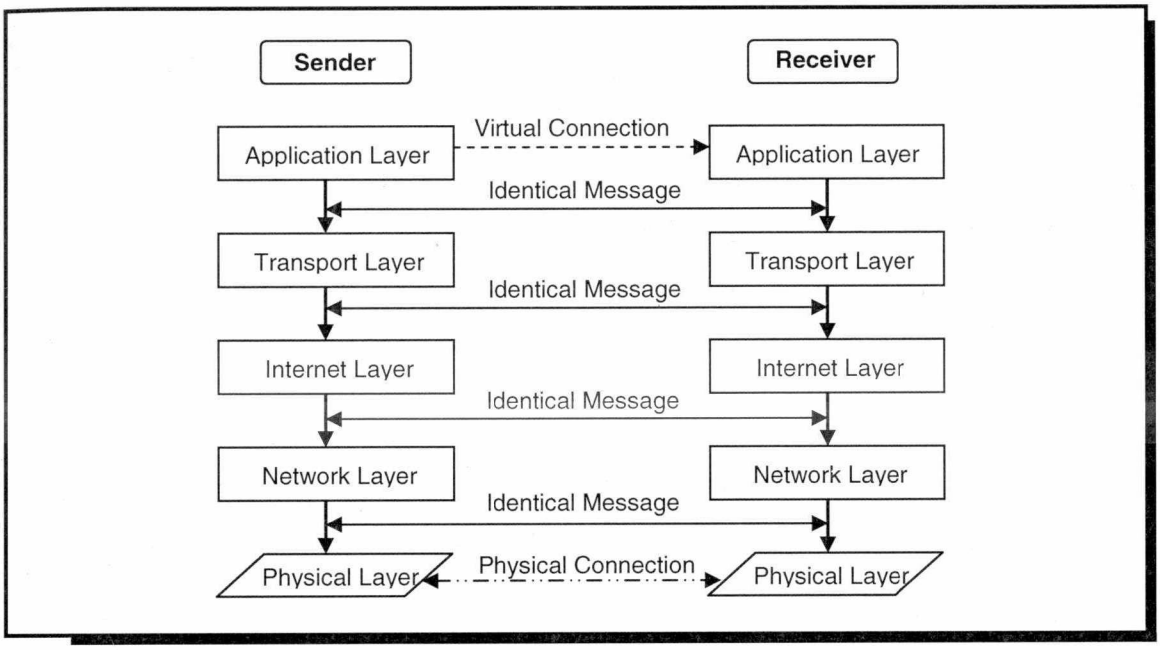


Figure 2-2: Data flow between two computers using TCP/IP stacks [2]

The name TCP/IP is misleading as TCP and IP are only two of many protocols that make up the TCP/IP suite; the term TCP/IP not only refers to the TCP and IP, but to the whole family of protocols associated to TCP/IP. The following sections explain each layer of the five-layer TCP/IP protocol stack.

2.2.1 Network Access Layer

The network access layer is the lowest layer in the TCP/IP Internet reference model. The function of this layer is to implement host specific transmission of datagrams; it is concerned with getting the data across a specific type of physical network. A few examples of types of physical network include: Ethernet (IEEE 802.3), token ring, wireless networks, etc. The responsibilities of the network layer include encapsulation of IP datagrams into frames to transmit over the network, providing control information that helps the route through the network, providing error checking information (Cyclic Redundancy Check) to outgoing frames for error detection at the receiver, and checking for errors in the incoming frames.

Unlike the other network protocol layers the network access layer must have the knowledge of the underlying physical network such as the frame structure, maximum frame size, and addressing scheme to ensure that the protocol formats the data correctly and transmits it over the network. In recent times, Ethernet is the most common physical network adopted in a local area network. The IEEE standard 802.3 defines the possible bit rates, addressing scheme, frame size, and format for the Ethernet. Table 2-3 shows the fields in an IEEE Ethernet frame.

Table 2-3: Fields in IEEE standard 802.3 (Ethernet) frame

Field	**Length in Bytes**	**Purpose**
Preamble	7	Synchronizing bits
Start of frame delimiter	1	End of synchronization pattern
Destination address	6	Destination Ethernet hardware address
Source address	6	Source Ethernet hardware address
Type or length	2	If the number of data bytes is less than or equal to 1500 it specifies length; If the number of data bytes is greater than or equal to 1536 it specifies the type
Data	46 to 1500	The data payload
Frame check sequence	4	Error checking value

2.2.2 Internet Layer

The network access layer adds header fields to the outgoing datagram that includes its own physical address and the destination address to locate the remote host on the network. This physical addressing scheme works well in an individual LAN segment with only few computers and interrupted media. However, it is not possible to deliver data on a routed network using the physical address of the destination. Even if it could deliver the packet, the process would be highly complicated, as there is no logical structure that can be imposed on the physical addressing scheme. TCP/IP, therefore,

organizes the network over a more logically structured addressing scheme using the Internet Protocol (IP) in the Internet layer.

2.2.2.1 Internet Protocol

Internet Protocol [3] is a packet-based connectionless protocol that does not require control information to establish the end-to-end connection before the data transmission. IP relies on the protocols from other layers if a connection oriented service is required. IP is also called an unreliable protocol, because it has to rely on protocols from other layers to provide error detection and error recovery schemes. The functions of the Internet Protocol include: preparation of IP datagram, defining the Internet addressing scheme, moving data between the transport layer and network access layer, routing datagrams to remote hosts, and fragmentation and reassembling of datagrams. The IP header includes the IP address of source and destination computers, the length of datagram, IP version and special instructions to the router.

IP addressing: IPv4 uses a 32 bit address called the Internet protocol address to identify a host on the network; it is different from the 48 bit Ethernet physical address specified by the network access layer. For routing efficiency, the IP address is defined by two parts: the prefix identifies the unique network in the Internet and the suffix identifies the host on the specified network. The Internet Assigned Number Authority (IANA) organizes the network numbers for the Internet. The number of bits allocated for the prefix and suffix in the IP address is specified by the class of IP address. Table 2-4 shows the description of classes, number of networks and number of hosts per network.

Net masks are used for identifying the number of bits in the prefix to extract network ID and to identify the host for a specified network. The network ID is identified by performing logical bit-wise-AND operation of the IP address and net mask. The net mask for class A networks is always 255.0.0.0, for class B networks it is 255.255.0.0, and for class C the net mask is 255.255.255.0.

The IP next generation (IPng), popularly called Internet Protocol version 6 (IPv6), is a modified version of IPv4. The main improvement brought by IPv6 over IPv4 is the increase in number of IP addresses available for network devices. Unlike IPv4, which can

support about 4.3 billion (2^{32}) addresses, IPv6 with its larger address space can support about 2^{128} addresses. Similar to IPv4, IPv6 is defined by two parts: a 64-bit prefix which identifies the unique network on the Internet and a 64-bit suffix which identifies the host on the specified network. Only a small part of the Internet uses the IPv6, and the full transition to IPv6 is assumed to take at least a decade.

Table 2-4: Classification of IP address with respect to its classes

Class	First 4 bits of IP address	Number of prefix bits	Maximum number of networks	Number of suffix bits	Maximum number of hosts per network
A	0xxx	7	128 (2^7)	24	16,777,216(2^{24})
B	10xx	14	16,384 (2^{14})	16	65,536 (2^{16})
C	110x	21	2,097,152 (2^{21})	8	256 (2^8)
D	1110	Multicast			
E	1111	Reserved for future use			

2.2.2.2 Address Resolution Protocol (ARP)

ARP [4] is used to determine the physical network address (Example: Ethernet / MAC address) from a logical or virtual address (Example: IP address). The host must know the physical address of the destination network adaptor in order to send any data, but the host network hardware does not understand the software implemented logical address. Hence, IP uses ARP to translate the virtual address to physical address.

2.2.2.3 Internet Control Message Protocol (ICMP)

ICMP [5] is the part of Internet layer and uses the datagram services of the Internet protocol to send its messages. The functions of ICMP include flow control, redirecting routes, checking remote hosts, and detecting unreachable hosts.

2.2.3 Transport Layer

The Internet layer provides the routing and addressing schemes so that the data can make their journey across the network, but the TCP/IP suite requires an additional layer to help the data reach their destination effectively without errors. The functions of the transport layer include error checking, flow control, verification, and an interface for network applications. The transport layer provides two different data paths to the network with different approaches to quality assurance; the two protocols are Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). Applications that need an interaction session and high reliability use TCP, and applications that do their own error checking or the applications that do not need much error checking uses UDP.

2.2.3.1 Transmission Control Protocol

TCP is a connection oriented, host-to-host protocol used in packet-switched networks and it provides extensive error control and flow control mechanisms for reliable data transfer. The services provided by TCP include: [6]

- *Basic data transfer:* TCP processes data in a stream of bytes in segments and communicates through IP frames.
- *Reliability:* The 32 bit sequence number and acknowledgement number provides an increased security of data transfer. If the frames are received out of order, the sequence number helps the re-sequencing of TCP frames, and the acknowledgement number confirms the received message. If the frame is lost, damaged, or duplicated, the sequence number helps to retransmit the lost message again.
- *Flow control:* TCP's flow control feature ensures that the data is not overrunning the destination machine's receiving capability. Each TCP frame gives the buffer the handling capabilities of the receiving machine.
- *Connections:* The TCP adds the control information, header and/or trailer that carry necessary instructions to establish a reliable data transfer between the sender and receiver. This is done by a three-way handshake.

- *Multiplexing:* A predefined internal address that serves as the pathway from the application layer to the transport layer is called a port. The ports are used by the TCP to allow multiplexing of IP addresses.

2.2.3.2 User Datagram Protocol

UDP [7] is a connectionless transport protocol, much simpler than TCP, and it does not perform any prior connection establishing sequence for data transfer. Unlike TCP, UDP does not support error recovery and flow control techniques. UDP adds the control information header, which includes the source and destination port numbers, length of data, and checksum. As UDP does not include source and destination IP addresses, it is possible that the datagram is delivered to the wrong address, but the connectionless design makes the UDP a choice for network broadcast situations.

2.2.4 Application Layer

The application layer is the top layer in the TCP/IP protocol suite. It provides the data to be sent on the network and uses the data received from the network. The seven layer OSI model has been very influential in the development of five-layer TCP/IP mode. The application layer, presentation layer, and session layer of the OSI model were combined into a single layer called application layer in the TCP/IP model. The description of these layers from the OSI model that corresponds to TCP/IP's application layer is as follows: [8]

- *Application Layer:* The OSI model's application layer defines protocols for applications that use the network.
- *Presentation Layer:* The presentation layer handles encryption and data compression functions.
- *Session Layer:* The session layer manages communication sessions between applications on networked computers.

The functions of the application layer are to send and receive data over the network; the data can be a single byte, line of text, or request for a web page. The data sent by the applications follow a protocol and can use a standard protocol such as HTTP

for requesting and sending web pages, File Transfer Protocol (FTP) for transferring files, or Simple Mail Transfer Protocol (SMTP) or Post Office Protocol (POP3) for sending emails.

The application layer has many TCP/IP utilities which are used to configure, monitor and troubleshoot TCP/IP networks. Versions of these utilities are default applications in many operation systems. Some examples of application layer utility programs are IPconfig, Ping, Netstat, and Hostname.

2.3 Embedded Web Server

Web-enabling a network device helps the user to access the device using any standard web browser, independent of the operating system. This eliminates the need for a new client software. The standard web browser by itself is a highly sophisticated application which presents the status of the device and helps to monitor the device from anywhere. An embedded web server can be implemented in very small footprints making it compact and portable, and allows easy integration into any device.

Embedded web servers are different from general web servers in many aspects. These differences can be observed in both functional and non-functional requirements. The functional requirements of the embedded web servers include handling HTTP packets, decoding the requests, and providing server side programming using limited resources. The non-functional requirements of embedded web servers include making the server a lightweight version of a general web server, making it compact and portable, and working with very low resource utility.

2.3.1 General Web Servers and Embedded Web Servers

A web server, also referred to as an HTTP server, is a computer program that listens to the requests from the web browser for documents typically stored on a server disk. The server then retrieves the requested document and responds to the browser along with the identifying information, for example, the type and size of the document. The web browser can display HTML documents directly, or use the identifying information to select the configured program to show the document.

To request a document from the server, the browser has to address the document using the Universal Resource Locator (URL). The standard form of URL looks like "protocol://server/request-URI." The protocol here defines the governing rules used for the communication between the browser and the server. The protocol is usually HTTP. HTTP is a client driven protocol, which means that the communication is always initiated by the web client. Once the connection is established, the web server expects one of the three basic requesting commands (GET, HEAD and POST) along with the document path specified in URL. GET is the most common form of request and basically means "send me this particular document." The request is sent by the client using GET in the form "GET document_path HTTP/version." The browser can include a number of header fields apart from a simple GET statement to give the server more information about the request. Some of the header fields that can be used with a GET command include user agent, referrer, from, authorization, etc. The browser can send the additional information related to the request by adding it at the end of URL. Another form of request that can be sent to the server is HEAD. The request HEAD works exactly as GET but the server responds with only the header for a particular document without actually downloading the document. HEAD is used by programs like link checker, to study and verify response headers. The third form of request, POST, is used to push the data into the server using forms. POST contains both a header and a body. The data encoded from the forms are sent in the body of the request. POST is used over GET if the information to be sent is longer than 256 characters. It is risky to use GET for passing information longer than 256 characters, as the URL is truncated by some operating systems during the request. Table 2-5 summarizes the working of important HTTP request methods.

The Uniform Resource Locator addresses the firmware and the server responds with the appropriate HTML web document. The HTML page is made up of web buttons, dropdown lists, hot links, regular text content, etc. The user enters his or her choice of inputs and sends the request back to the server. The web server then performs the required operation according to the request. The operation can be: sending remote commands to another device that is connected to the server, collecting the information

from the device, or maintaining a queue of commands to be executed in the near future. After processing the requested command, the server sends the response to the browser for displaying the result on the screen. Figure 2-3 shows the request and response data flow between the browser and the web server using HTTP.

Table 2-5: Important HTTP request methods

Method	**Description**
GET	The client requests the resources by adding the information at the end of URL and the server sends the header fields followed by the requested resource
HEAD	Similar to GET, but the server sends only the header fields and not the resource itself
POST	The client requests the resources by sending resource information to the server, as in the case of pushing a button

The server processes the received HTTP request and returns the appropriate document. The first line of the response shows the protocol and the version used along with the HTTP status code and user recognizable status message. This is followed by several header fields which include: server date, last modified date, content length in bytes, content type, etc. The requested document is sent after all the header fields. The browser then performs the difficult task of presenting the text and graphics, including images, sounds and video.

A web server can be embedded into a device by reducing the resource requirements of the general web server. This extends the functionality of the user interface to the device and enables remote access to the device through the Internet using any standard web browser. These reduced resource requirements can be achieved by limiting the functionality of the full-fledged web server both in size and network handling capabilities. This type of web server is called an embedded web server. Integrating the web documents and images which hold the status information of the device and representing this information using graphics directly results in web-based user

management interface. The major role of an embedded web server is to listen to the requests from the web browser or web client and provide the device managed data to the web browser for monitoring and controlling the network device.

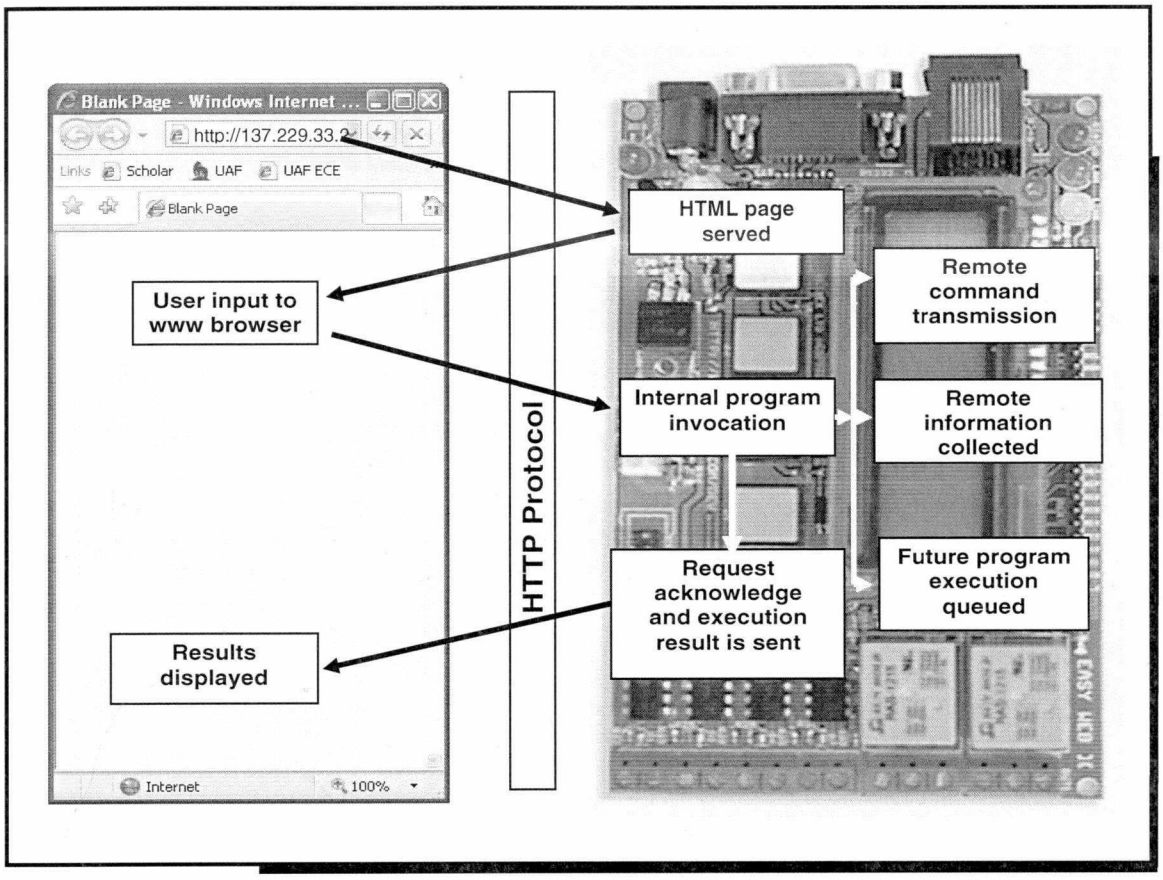


Figure 2-3: Request and response data flow between the browser and the web server using HTTP

2.3.2 Advantages of an Embedded Web Server

There are many reasons for incorporating a web server into a device. The graphical, user friendly, platform independent and universally available free web browser as a remote control for network device is increasing the demand for web-enabling a device. Following are the advantages for incorporating a web server into an embedded device: [9]

- *Eliminate implementing the target software for remotely accessing the device:* Adding remote access interface to a device essentially does not require target side

programming, as the device can be accessed through a universally available standard web browser.

- *Universally available clients:* The client software is platform independent and can be used with any operating system such as Windows, Linux, and Macintosh etc. The client software is available as a package with operating system and does not need any separate installation.
- *Remote or local access:* If the device is connected to the Internet, it can be accessed from any part of the world. The device can also be connected to local area network very easily for local access.
- *Easily create graphical user interface:* The device main web page provides central navigation point to the device controls. These web pages can be easily created using basic HTML programming and can be supplemented with many graphical interface features.
- *Commercially available web designing tools:* Apart from basic HTML programming and adding a graphical user interface (GUI), we can also enhance the web page design using more advanced web designing and authoring tools like Macromedia® Dreamweaver® or Microsoft® Front Page that are readily available on the market. The web pages can be designed using the web design tools well in advance and can be integrated into the device firmware with ease.
- *Easy to maintain and enhance interface to target:* The device web pages can be modified with enhanced interface to the device controls and can be easily updated to the device firmware.
- *Uses universally accepted, standard communication protocol:* There is no need to develop the communication protocol for accessing the device. The existing standard and universally accepted communication protocols are used for accessing the device.

2.3.3 Functional Requirements of an Embedded Web Server

The implementation of an embedded web server requires several functional requirements to be considered. These functional requirements include HTTP compatibility, embedding dynamic content into the device web page, integrating non-volatile memory into the device, and security for providing the administrative privileges. The two core functional requirements necessary for web-enabling a device are embedding the HTTP server into the device to handle the HTTP protocol used in the World Wide Web and the software mechanism for embedding the dynamic content into the HTML.

The basic function of an embedded web server is to handle the HTTP traffic. As in a general web server, the embedded web server has to listen to HTTP requests from the web client and respond with the data monitored by the network device. HTTP is a universally accepted standard protocol for communication between the server and the browser. It is basically a series of HTTP messages sent back and forth between the server and the browser. Implementing the HTTP is the easy part; the huge availability of embedded web servers in the market stands as an example for programming an embedded web server easily [10].

The response of the web server can have either static or dynamic content in the web page. The majority of general web servers still host only static content web pages. When a document is requested, the general web server retrieves the static web page and passes the information to the browser. Unlike the general web servers, the majority of embedded web servers host dynamic content in web documents. When a document is requested by the web browser, the embedded web server embeds the dynamic content into the web page and passes the information to the browser. The dynamic content can be in the form of the present state of the device or the data collected by the device.

The function of an embedded web server is to monitor and control the operation of the sensor network and to periodically collect the data from the sensor nodes. The embedded web server has to store the collected data in the nonvolatile memory (ROM) for later network data analysis. This requires an increased memory on the embedded web

server. Even the device web documents reside in the ROM. When a web document is requested, the embedded web server copies the web document into the random access memory (RAM) and passes the information to the browser after replacing the dynamic content in the web page.

The other important function of concern is the security in maintaining the device configuration and monitoring data. The embedded web server has to provide administrative privileges to permit access and control to the device data only to authorized users.

2.3.4 Non-functional Requirements of an Embedded Web Server

Apart from functional requirements of embedded web server design, which explain the working principle of the server, the design also has to consider the hardware or non-functional requirements of the server. Compared to general web servers, an embedded web server has very limited resources available. This creates a need to revaluate the functional issues of the embedded web server that affect the hardware requirements and to adopt optimum procedures to restrict the usage of resources without hindering the core functionality of the server.

Among the resource scarcity, memory is the primary restraint of the embedded web server; the embedded web server has to use its memory in the best possible way to accommodate all the primary functions of the server. The primary functions include handling the HTTP requests and responses, transportation and routing of messages through the Ethernet to reach the web client. The transportation and routing protocols used in handing the World Wide Web traffic are TCP/IP.

Due to limitations on memory usage the TCP/IP stack should be implemented with very limited functions. In other words, the memory usage can be reduced by limiting the network handling capabilities. Limiting the functions of network protocol for conserving memory takes drastic shortcuts in TCP/IP implementation, potentially crippling the network protocol. These shortcuts can come in the form of not implementing the parts of the protocol that are not significant to the application. An example of limiting the functionality can be dropping the UDP when the device uses only

TCP. The layer which suffers the most with the reduced functionalities in the International Standard Organization (ISO) reference model is the transport layer, and this layer usually contributes a large portion of the code size.

The general web server has its way to handle multiple requests from different web clients simultaneously. The multithreaded architecture of the general web server makes it possible to handle multiple requests simultaneously. Embedding this feature into an embedded web server is usually impractical due to memory overhead required and, in some cases, to the lack of embedded software support for multiple processes. The embedded web server handles a single request at a time and processes the multiple requests sequentially on a first-come, first-served basis. Limiting the number of simultaneous connections greatly reduces the data and program memory requirements.

Despite all the assumptions, the embedded web server is expected to perform its core functional requirements of handling the HTTP traffic and providing dynamic web content. However, these limitations impair the network handling capabilities of the embedded web server.

3 Related Studies

3.1 Introduction

The latest emerging embedded technologies are making web-enabled devices much easier to design and use. Firstly, to add the functionality of remote access to a network device, the HTTP server has to be incorporated into the device; the HTTP server handles the HTTP traffic and responds to the HTTP requests from the web client. HTTP is the universally accepted standard protocol, and its implementation is relatively straightforward. The bigger challenge is to embed the actual web documents into the server and handle the dynamic changes in the web content.

General web servers which operate on UNIX or Windows NT platforms have a large file system which is backed up with large hard disks. The web documents usually reside on these disks. Unlike general web servers, most embedded web servers do not have a file system. All the web documents have to be stored in the flash memory, together with the source code. There are many ways to embed web documents directly into server's source code. A simple approach could be writing HTML code in a normal text editor using basic HTML tags and incorporating the whole text into the server code using a single character buffer. The server code has the routines and functions defined which send the complete character buffer to the web browser as a web document. This approach can serve static web pages to the browser easily, but with the inclusion of dynamic data content into the string, the program has to adopt a slightly more complex algorithm. The dynamic data is identified using a unique embedded tag included into the static web page. A small routine will search through the entire character buffer and replace the embedded tags with the appropriate data before sending the contents of the character buffer to the browser.

With the increased complexity of web design, the web designer no longer writes the HTML code in a normal text editor; instead he uses a web designing tool to generate rich graphic web documents. Since the static HTML page constitutes only a small part of

the web content, the dynamic content is of great importance and no web designing tool will solve the problem of providing dynamic content.

To overcome this limitation, embedded system designers are developing different service routines to automatically convert the web content into the server's source code. These functions insert non-HTML tags to represent the dynamic data. Later, a conversion tool recognizes these tags and initiates necessary functions. After the appropriate dynamic content is returned, the same conversion tool is responsible for updating the non-HTML tags with the dynamic content. The above discussed techniques are used by most embedded web servers to handle the dynamic content in a web page and different products have opted different algorithms to achieve this. The following sections discuss the research projects and several commercially available embedded web servers, their features, advantages and limitations.

3.2 Research Projects

Active research is carried out in the field of embedded web servers, and many new techniques and features are explored every day to easily integrate the embedded web server into any electronic device and appliance. This section discusses some of the embedded web server research projects which explain the implementation of HTTP server and their capabilities.

3.2.1 POSTECH - POS-EWS

The POStech – Embedded Web Server (POS-EWS) [11] from Phang University of Science and Technology is an HTTP/1.1 compliant embedded web server implemented on the Xinu OS using the MPC 860 processor. The POS-EWS is implemented with code space of around 30 KB. The POS-EWS architecture consists of five parts: HTTP engine, an application interface module, a virtual file system, a configuration module and a security module.

The HTTP engine implements the subset of the HTTP features typically required for an embedded web server and handles the HTTP requests from the web client. The HTTP engine also includes cache control and support for persistent TCP connections

required by HTTP/1.1. The application interface module enables the developers to add new management functionalities. POS-EWS supports both Common Gateway Interface (CGI) and Server Side Include (SSI) protocols to process the user input and HTML form data. The virtual file system provides POS-EWS with virtual file services, and it uses a web complier to compile the web documents into intermediate C code and links them to the server code. The web compiler can also compress the web documents to conserve memory and serve the compressed documents along with the accept-encoding and content-encoding header fields to the web browser. HTTP/1.1 can convey the compressed documents along with header fields and the web browser decompresses the document to present them to the client. The security and configuration module provides basic and digest authentication and encryption over the HTTP network.

3.2.2 BS2P40

Polytechnic University, Brooklyn developed an embedded web server that controls the DC motor position [12]. The embedded web server hosts a web page with the controls of the DC motor. The controls include a text box and a slider interface developed using java applets that commands the motor position and allows values from 0 to 360 degrees. The output is graphically represented on the web page and displays the current position of the motor using the real time sensor data from the web server. Unlike many embedded web servers that use TCP to establish the communication between the client and server, the server developed by Polytechnic University uses UDP to communicate with the client and is implemented in PBasic.

The embedded web server is a 10Base-T Ethernet board manufactured by NetBurner [13] and connects the BS2P40 microcontroller to Ethernet. The Basic Stamp 2 (BS2P40) microcontroller manufactured by Parallax, Inc. [14] is a 40 pin Dual Inline Package (DIP) with ROM. It has 16 KB of Electronically Erasable Programmable Read Only Memory (EEPROM) and small RAM capacity, and it is used to drive the DC motor.

The DC motor setup includes an armature controlled DC motor, a continuous rotation potentiometer, a rotary optical encoder, a tachometer, and a power amplifier. The potentiometer provides the angular position measurement of the DC motor; it outputs a

voltage signal related to the DC angular position within a range of +/-5 VDC. The BS2P40 controls the angular position of the DC motor by applying a controlled voltage signal.

The embedded web server continuously listens for UDP packets intended for the DC motor. When the server receives a request, it extracts the user defined command angle from the UDP packet and forwards it to the BS2P40 microcontroller. The BS2P40 computes the control signal to the DC motor using the current motor position and user defined angle. The microcontroller then issues the commands to the DC motor to change the motor position to the user desired value, and also returns the summary back to the server to report the results to the client.

3.2.3 Hardware-In-the-Loop Simulator (HILS)

Researchers from the University of Alaska, Fairbanks developed an Internet-based remotely accessible Hardware-In-the-Loop Simulator (HILS) [15] for robotics and mechatronics applications.

The HILS setup includes two motors coupled to each other; one of the motors represents a joint actuator for the robot arm and another motor acts as a load simulator. The setup also includes the DS1104 microcontroller board, a product of dSPACE, which is used as a motor control unit. This card is connected to a server to establish a connection between the remote client and the HILS. The client and server software is developed using wxWidgets. The client software controls and tracks the position of any robotic arm. The MATLAB® Engine is used to download the Simulink files to the DS1104, which in turn drives the HILS. The TCP/IP is used to establish the connection between the remote client and the server. The simulation results are sent to the remote user in the form of .mat and .jpeg files.

3.2.4 Hewlett Packard - Cooltown

The Hewlett Packard Laboratory – Cooltown [16] project is exploring the convergence of web technology, wireless networks, and portable client devices through an infrastructure to support "web presence" for people, places, and things. This project is

pushing web technology into the digital appliances or things like printers, radios, automobiles, etc., and explores different ways for digital communication appliances to interact with these web-enabled devices. Cooltown wants to make people, places and things web present: the things can be made web present by embedding web server into the device, the places can be made web present by managing web things into a single collection through a web service called "PlaceManager," and people can become web present by offering information through location specific PlaceManagers.

The project extends the capability of interacting with a web-enabled device through a wireless interface. Many digital communication appliances provide this wireless interface with small efforts by making appropriate extensions to web technology. Appliances like Portable Digital Assistants (PDA), laptops, watches, etc., can be used to control and monitor the web-enabled device. The digital appliance can receive the URLs from the wireless beacon, which is a small infrared transceiver placed near the web-enabled device. The URL links to the device home page through which it can be controlled and monitored. An alternative approach to accessing the device home page can be by resolving the tag placed near the device with the help of a gateway web server and then receiving the URL that links to the device home page.

3.2.5 VTT Electronics – WebChip

Most of the embedded Internet appliances use a microcontroller and software to enable TCP/IP and support HTTP, but researchers from University of Oulu, Finland and VTT Electronics argue that there are many appliances where a complete hardware based approach is more suitable. They developed WebChip [17], a TCP/IPv6 and HTTP/1.1 compatible, functionally minimized embedded web server implemented with a C code library and VHDL code. The solution is tested using Field Programmable Gate Array (FPGA) and can be later integrated into various Application Specific Integrated Circuits (ASIC).

WebChip is implemented with several minimizations to the IPv6 protocol. The IPv6 headers are ignored to reduce the complexity of IPv6 part of protocol, with the exception of authentication and encapsulating security. WebChip responds to two kinds

of network traffic. It responds to the queries from neighboring appliances for link layer address (Media Access Control address) of the WebChip, and also responds to the HTTP requests from the web client.

The hardware implementation of WebChip consists of IPv6 packet filter, TCP connection handler, TCP connection timer, ICMPv6 protocol interpreter, HTTP memory, HTTP/1.1 interpreter and home page memory. IPv6 packet filter checks the packet for the server and validates the packets and routes to the appropriate block after removing all IP header fields. The TCP connection handler and TCP connection timer keeps track of TCP connection status and checks a timer to reset the connection at timeout. The HTTP protocol interpreter handles the HTTP traffic using HTTP memory to store the HTTP header information. The hardware is designed using synthesizable RTL – VHDL. The project was demonstrated using an Altera FPGA (APEX 2K100) and an Ethernet controller. The amount of logics is less than 10,000 gates; the server has 4 Kbits of memory for storing the device web home page and HTTP header.

3.3 Commercially Available Embedded Web Servers

This section deals with the survey of commercially available embedded web servers their supporting features, and techniques adopted to handle the dynamic web pages.

3.3.1 Blunk Microsystems – TargetWeb

TargetWeb [18] is designed to run on any 32-bit CPU or DSP architecture that has a C compiler and supports 8-, 16- and 32-bit access. HTTP versions 1.0 and 1.1 compatible, TargetWeb is a high performance embedded web server used to control and monitor the embedded applications using any standard browser.

Blunk Microsystems developed TargetOS, the Real Time Operating System designed specifically for embedded applications. TargetOS is integrated with TargetOsTools (the IDE for embedded development with an integrated complier and kernel aware debugger). TargetOsTools also includes TCP/IP protocol stack, TargetTCP, flash file system and LAPB protocol stack. TargetWeb is tightly integrated with the

TargetTCP; it uses TargetTCP's zero copy for high performance. The code space used for implementing TargetWeb is around 30 KB and 30 KB of data space to the application.

TargetWeb uses a simple mechanism for generating dynamic HTML pages; it embeds custom <DATA> tags into HTML pages allowing the server to replace the tags with the dynamic content. HTML pages on TargetWeb are easy to configure and maintain. HTML pages can be updated using TargetTCP's inbuilt File Transfer Protocol (FTP) server. Another feature of TargetWeb is compression of static file system using TargetZFS. Like all other TargetOsTools, TargetZFS is also a royalty free feature coded in ANSI C.

3.3.2 GoAhead Software – Web Server 2.1

GoAhead web server [19] is a royalty free, open scripting architecture, which is a fully functional standard based embedded web server. GoAhead web server 2.1 distributions include source code written in C and reference platforms for Windows, Linux, Lynx, QNX, and eCos. The HTTP 1.0 and HTTP 1.1 compliant web server 2.1 provides support for dynamic content with Active Server Pages (ASP), in-process CGI, traditional CGI, and Embedded JavaScript. ASP is the standard developed by Microsoft for supporting dynamic web content. Along with Embedded JavaScript, a strict subset of JavaScript, ASP provides highly efficient dynamic web content with much smaller web page size.

GoAhead web server 2.1 also includes support for Secure Socket Layer (SSL) for authentication and encryption over TCP/IP networks, and Digest Access Authentication (DAA) for password encryption, to provide more secure authentication. The other features of the server 2.1 include support for different levels of user access using User Management functionality of the server. The memory footprint of web server 2.1 is less than 60 KB of code.

3.3.3 Mbedthis - AppWeb HTTP Server

AppWeb [20] is a standard HTTP server which supports HTTP 1.1 and CGI 1.1. The key features of the web server include easy dynamic page creation using Embedded

JavaScript, Embedded Server Pages, Embedded Gateway Interface, and CGI. Embedded Server Pages are server-side programming applications written using Java procedures and logic routines of the application. Embedded Gateway Interface is an alternative to CGI for data processing. It is an equivalent to in-process CGI. Embedded Gateway Interface supports both static and dynamic web pages. AppWeb supports modular architecture; the software dynamically selects only the features that are needed by the application.

AppWeb closely guards the system resources to increase the reliability of the system through a technique called "sandboxing." AppWeb defines the directives for the HTTP server to work in a controlled environment. The controls can be in the form of rejecting requests and URLs that are too long, pre-allocating memory and ensuring the restricted use of memory, etc.

AppWeb supports SSL, basic and digest authentication to enable restricted access to the web server's resources over the TCP/IP network. The memory footprint of AppWeb is around 110 KB of code.

3.3.4 AllegroSoft – RomPager

RomPager [21] was designed to support devices with ROM based real time environments. The RomPager embedded web server toolkit includes HTTP 1.0 and HTTP 1.1 server, compatible with all standard browsers, and supports different levels of HTML (level 2.0, 3.1, and 4.0) with an intelligent HTML web page compression technique for efficient use of memory. RomPager also supports features like dynamic HTML creation, efficient internationalization (supports standard error messages in various languages such as French, German, English, etc), flexible security, and built-in numeric conversion routines. RomPager also comes with the optional features that include file system, remote host support, email notifications, and email message reading.

RomPager also comes with advanced embedded web server toolkit, which includes multiple web object sources, object compression, and advanced security. The advanced toolkit also includes a web application toolkit and page builder compiler. The appealing feature of the RomPager web server is the PageBuilder compiler which is used to design web pages hosted by the server. The PageBuilder compiler takes the web pages

designed by any web designing tool as an input, and generates a highly compressed web object that is later complied and linked to the main project.

RomPager also includes another interesting toolkit called the Remote Host toolkit. It is a special purpose web proxy which stores large files on another web server and serves them through the embedded device as if they are stored on the device itself.

3.3.5 BVM – IntraScada Web Server

IntraScada web server [22] is designed for OS-9 systems and is compatible with any 680x0 machine, from single board controllers using CPU32 to 68060 processors. The code space required for the web server is less than 100 KB and the data space is 20 KB, plus 28 KB for each connection. The device does not necessarily need to have a file storage system. If the device has a file storage system, HTML pages and graphic images are served from it, or else the HTML pages and graphics can be retrieved from elsewhere on the network and these pages can be served by the server after embedding the dynamic content. The key features of the IntraScada web server include the compatibility with HTTP 1.1 and HTTP 1.0 persistent connections, and it supports CGI for the fastest possible response.

Real time operating systems such as OS-9 are reportedly good at controlling processes, but they are not suitable for presenting the graphic rich text. Such displays are possible using the Scada software suite. However, this solution comes with numerous restrictions. The software resultant from the Scada suite has no portability and is specific to a particular installation.

3.3.6 Quiotix – Quiotix Embedded Web Server (QEWS)

Quiotix consider their QEWS [23] to be the world's first modular web server. The standard QEWS distribution includes system dependent modules for Microsoft Windows, Sun Solaris, DEC UNIX and popular embedded operating systems such as Lynx and VxWorks.

The features of QEWS include full HTTP support and support for internal CGI 1.1 and Server Side Include (SSI). The security mechanisms include basic and digest

authentication. QWES also supports virtual file system with web page compression technique to minimize the memory footprint. Another feature of QEWS is a preprocessor/HTML compiler. By using this compiler, QEWS automatically generates calls/entry points to extract input values from forms by assigning handlers to URLs in the virtual file system. The entire application is then included in the main program which runs on the target device.

The interesting feature of Quiotix is its exclusive AutoSubset technology. Quiotix automatically configures itself to generate a minimal subset for each application. Unlike other embedded web server designs which are based on minimalist approach, QEWS supports full HTTP server features but includes only those server subsets which are used by the application.

3.3.7 Moteiv – Tmote Connect

Tmote™ Connect from Moteiv is a wireless gateway application used to access Tmote wireless sensor modules through Ethernet. The wireless sensor modules are connected to the wired local area network using a modified Linksys NSLU2, a network attached storage device. Tmote Connect software programs the Linksys NSLU2 to function as a gateway to connect the wireless sensor modules to the Ethernet. Each Tmote wireless module can be remotely monitored and controlled through a web-based graphical user interface.

Tmote Connect supports up to two Tmote wireless modules and all the sensor network administrative controls can be accessed through a single web page interface running on port 80. The web page presents the information corresponding to individual Tmote Connect devices. The web page presents the device information, number of clients in the network, number of packets read and written, the serial forwarder and control port numbers. The administrative tasks controlled through the web page include: individual motes reset, restart the server programs, and reboot Tmote Connect. The key features of Tmote Connect software include: [24]

- Two-way data transfer handling capability between the client and the wireless sensor modules over the TCP/IP networks.

- Web-based graphical user interface used for mote identification, reset and performance counters.
- Support for DHCP and non-DHCP networks.
- Flash reprogramming of Tmote wireless sensor modules remotely using standard in-system programming protocols.

Unlike most embedded web servers which integrate themselves into the web-enabling device, Tmote Connect software enables the network attached storage device to function as a gateway to connect the wireless sensor network to the Ethernet. Tmote Connect software is a TCP/IP implementation specific to the Tmote wireless sensor network. Tmote Connect uses a single web page to control and monitor the sensor network and does not allow additional web pages and controls.

Table 3-1 compares the features of all the above discussed commercially available embedded web servers; a blank indicates a feature not supported or data not available. None of the solutions presented in Table 3-1 fulfils all the requirements for web-enabling a wireless sensor network. For example, the Quiotix embedded web server includes all the features presented in Table 3-1. However, because of its limited onboard memory, it is not suitable for data collecting. The Tmote Connect's functionalities are limited only to reprogramming or resetting the sensor nodes in the network.

This thesis presents two embedded web servers developed using different hardware platforms. One, based on a very low-power MSP430 microcontroller, has somewhat limited functionality but presents a viable solution for situations where energy efficiency and the total cost of the solution are of primary interest. The other solution uses a more powerful Rabbit RCM3750 microcontroller board and implements a fully-functional, dynamic web server with multiple web pages. Chapter 5 presents the limitations of the MSP430-based solution and all the features and capabilities of the Rabbit-based web server.

Table 3-1: Comparison of commercially available embedded web servers

Company and Product	OS Supported	CPU supported	HTTP code Size and Version	Features			
				SSI	VFS	Security	CGI
Blunk Microsystems - TargetWeb	TargetOS, RTOS	Any 32bit CPU with C compiler	30 KB, HTTP 1.1	X	X	Basic, Digest and SSL	
GoAhead Software - Web Server 2.1	Linux, Lynx, QNX, and eCos	Any CPU with C compiler	60 KB, HTTP 1.1	X		Basic, Digest and SSL	X
Mbedthis - AppWeb HTTP Server	Linux, VxWorks, Mac OSX, Solaris, Windows	ARM7, MIPS32, i386/X86, PowerPC, Sparc	110 KB, HTTP 1.1			Basic, Digest and SSL	X
AllegroSoft - RomPager	Any RTOS	Any CPU with ANCI C compiler	10 to 40 KB, HTTP 1.1	X	X	Basic and Digest	
BVM -IntraScada Web Server	OS-9	CPU32	100 KB, HTTP 1.1				X
Quiotix - QEWS	pSOS, LynxOS, VxWorks	Any CPU with C compiler	45 to 50 KB, HTTP 1.0	X	X	Basic and Digest	X

3.4 Embedded Web Server Applications

The WWW is connecting all the nations, societies and cultures together; the web is a cyber space with web sites, email, online malls, chat rooms, and many handy applications. WWW is the world's largest computer network; any device can be web-enabled and can be controlled from anywhere in the world through the web. A web server can be embedded into a device to extend its user interface options and expand its reach of control. This section gives a couple of examples of embedded web servers at work.

3.4.1 WebCoffee

The University of Delaware, Remote Access Instrumentation Laboratory (RAIL) designed a miniaturized, Linux based web server within a Capresso coffeemaker called WebCoffee [25]. Any Ethernet connected computer around the globe can brew coffee by a single click on the web coffee's home page and the user can even set the strength of coffee from strong to weak. WebCoffee is implemented using a standard embedded PC running a miniaturized combination of Linux kernel and a web server. WebCoffee is simple and small, and provides a web-based service to read information from web coffeemaker and switch it ON/OFF through a password control.

3.4.2 Ethernet Connected Sprinkler Controller

The sprinkler controller is a device used to electronically control a sprinkler to water the lawn. Embedding a microcontroller based embedded web server in the sprinkler controller extends the functionality of the device user interface and enables the controller to be controlled remotely via a PC or laptop using universally available web browser and without need for an additional software.

Rabbit Semiconductors has demonstrated the use of an embedded web server to remotely control the Ethernet connected sprinkler [26]. The aim of the project was to develop a cost efficient, Ethernet connected sprinkler. The microcontroller in the embedded web server is also configured to replace the existing controller and to make intelligent decisions to control the duration of watering on a rainy day or on a hot day. The controller can make a decision of either increasing the watering duration on a hot day or decreasing the duration on a rainy day using the temperature sensor to sense the day temperature and a humidity sensor to sense the moisture content in the atmosphere. The embedded web server also presents the intuitive interface to control the sprinkler online.

The Rabbit Semiconductor RCM3400 microcontroller is used to web-enable and control the sprinkler. The RCM3400, along with Wescor S-460 leaf probe humidity sensor and PT100 temperature sensor is used to control the four valve sprinkler. The embedded web server RCM3400 provides a very versatile user interface to control and monitor the sprinkler. The user interface allows the user to set the date and time of the

web server, set the sprinkler schedule, and check the current status of the sprinkler system. The clock web page is used to set the time, day, and date of the web server; the Real Time Clock (RTC) of the server is updated using this page. The current status web page presents the operation status of all four sprinklers along with daily weather forecast. The water schedule web page is made up of four sprinklers with a 24 hour duration table to set the time of watering for each of the four valves.

Once the user selects the schedule, the event table in the server is updated and using the RTC, simple start and stop commands are issued to engage the sprinkler and water the lawn for 10 minutes. The server also takes the user requests into account for monitoring the temperature and moisture content and updates the event table accordingly. The hot day rule states that if the temperature of the day is more than 100^{o} F, the watering duration is increased to 15 minutes instead of the normal 10 minute schedule. Similarly, the rainy day rule states that the watering duration should be decreased to 5 minutes on a rainy day. This means the sprinkler will be ON for 15 minutes on a hot day and only 5 minutes on a rainy day.

4 MSP430-based Web Server

4.1 Introduction

The aim of this thesis is to develop an embedded web server which is capable of monitoring, controlling, and collecting the data from the wireless sensor network. The embedded web server extends the monitoring and controlling capabilities of the wireless sensor network. The user will be able to monitor and control the wireless sensor network remotely and access the network data from any part of the world. The project setup includes the embedded web server connected to one of the wireless sensor nodes of the network and the wireless sensor network itself which monitors the network parameters including temperature, humidity, battery voltage of individual sensors, etc. This chapter describes the hardware required and software implemented to develop a fully-functional embedded web server that fulfills the above stated capabilities.

General web servers are designed to serve static web pages from a computer terminal with plenty of CPU resources. Embedded web servers, unlike general web servers, have different hardware design considerations. However, the final product is supposed to replicate the general web server functionalities such as handling HTTP requests and hosting web pages. The hardware architecture of an embedded web server should include a microcontroller to host the user application and its web pages and handle all the necessary computations, and an Ethernet module to connect the Internet. The design considerations include selection of a microcontroller with optimum volatile and non-volatile memory, and an Ethernet module that can easily interface with the microcontroller. The wireless sensor nodes in the network are implemented using the MSP430 microcontroller from Texas Instruments and this makes MSP430 a logical choice to implement the embedded web server. The application report [5] from the Texas Instrument on Internet connectivity using the MSP430 provided us a starting point towards the implementation of an embedded web server. Olimex Limited [27] developed an embedded web server using the MSP430 microcontroller. The following sections present the hardware and software implementation of the MSP430-based web server.

4.2 Hardware Description

The two main components used to develop the MSP430-based embedded web server are the MSP430F149 microcontroller from Texas Instruments and CS8900A Ethernet controller from Crystal™ Semiconductor Corporation. The following two sections describe the features of the above microcontrollers that make them a good choice for web server development.

4.2.1 MSP430F149

The MSP430 family is a broad family of ultra low power, RISC type, Von Neuman CPU core, 16-bit microcontrollers from Texas Instruments. Its architecture combined with five low power modes makes it a popular choice for low power applications. The current drawn when the MSP430 is in the lowest power consuming mode is in the order of few microamperes. The MSP430 family of devices ranges from very small memory (1 KB ROM, 128 B RAM), to large memory devices (60 KB ROM, 2 KB RAM). Figure 4-1 shows the complete Ethernet board solution. The major functional blocks include: 16-bit CPU, memory map, hardware multiplier, clock source, timer, Analog-to-Digital Converter (ADC), and USART.

MSP430F149's 60 KB of ROM and 2 KB of RAM make it a good choice for a web server. The MSP430F149 with 60 KB of flash memory can store and transfer web pages. The six general purpose input-output ports can not only interface with Ethernet controllers easily, but can also be used to realize a user project. The other features of MSP430F149 include: [28]

- *Digital I/O:* There are six 8-bit input-output ports, ports 1 to 6. All individual I/O ports are independently programmable, allowing 48 digital I/O pins.
- *Watchdog Timer:* The watchdog timer is used to perform controlled system restart on software error. The watchdog can also be used as a general purpose timer and can generate interrupts at selected time intervals.

- *Hardware multiplier:* MSP430F149 has a dedicated peripheral module and performs 16×16, 16×8, 8×16, and 8×8 signed and unsigned multiplication operations.
- *Universal Synchronous Asynchronous Receive Transmit (USART):* MSP430F149 supports two USART peripheral modules for serial data. The USART features include: 7- or 8-bit data with odd, even, or no parity, separate transmitter and receiver registers, Least Significant Byte (LSB) first data transmit and receive, and independent interrupt capability for transfer and receive.
- *Timer:* MSP430F149 includes two 16-bit timers with three capture/compare registers. Timers can support multiple capture/compare, Pulse Width Modulation (PWM) outputs, and interval timing.
- *Analog-to-Digital Converter (ADC):* The MSP430F149 has fast 12-bit ADC module. ADC12 features a maximum conversion rate of greater than 200 kilo-samples per second. The conversion can be initiated by software or timer, selectable conversion source, etc.

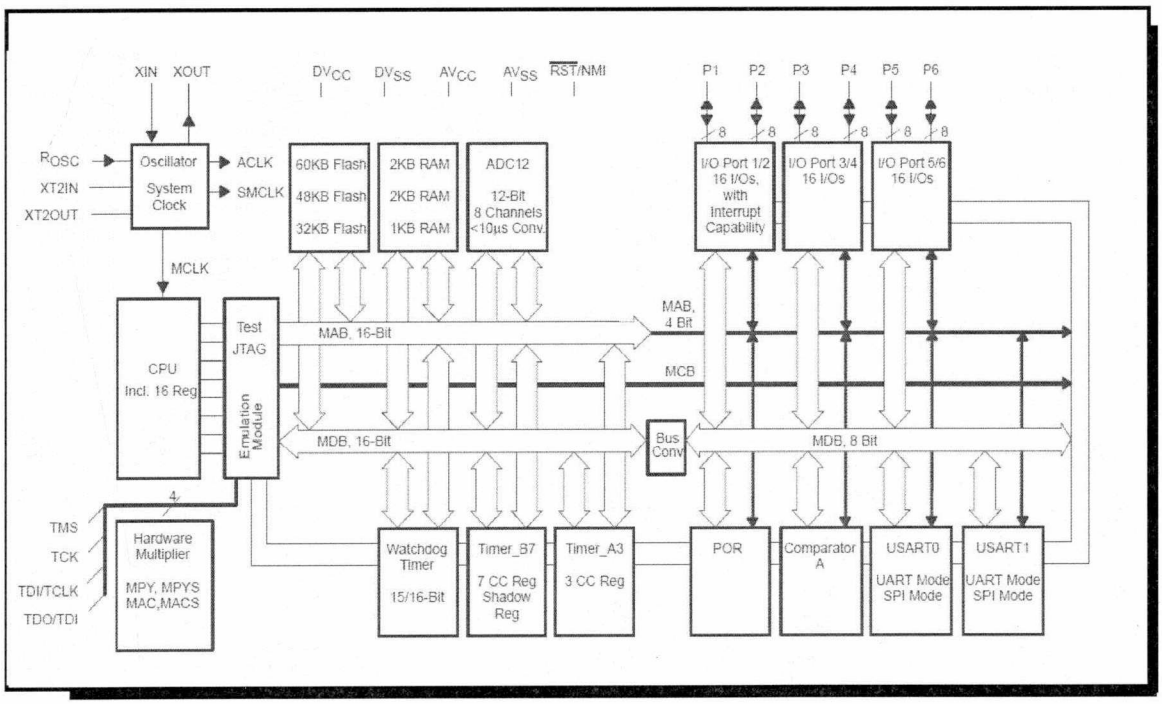


Figure 4-1: Functional block diagram of MSP430F149 [28]

4.2.2 CS8900A

CS8900A from Crystal™ Semiconductor Corporation is a low cost Ethernet LAN controller optimized for the Industrial Standard Architecture (ISA) bus and general purpose microcontroller buses. Figure 4-2 shows the complete Ethernet board solution. The major functional blocks include: a direct ISA bus interface, an 802.3 MAC engine, integrated buffer memory, and complete analog and digital circuitry for 10Base-T and AUI interfaces.

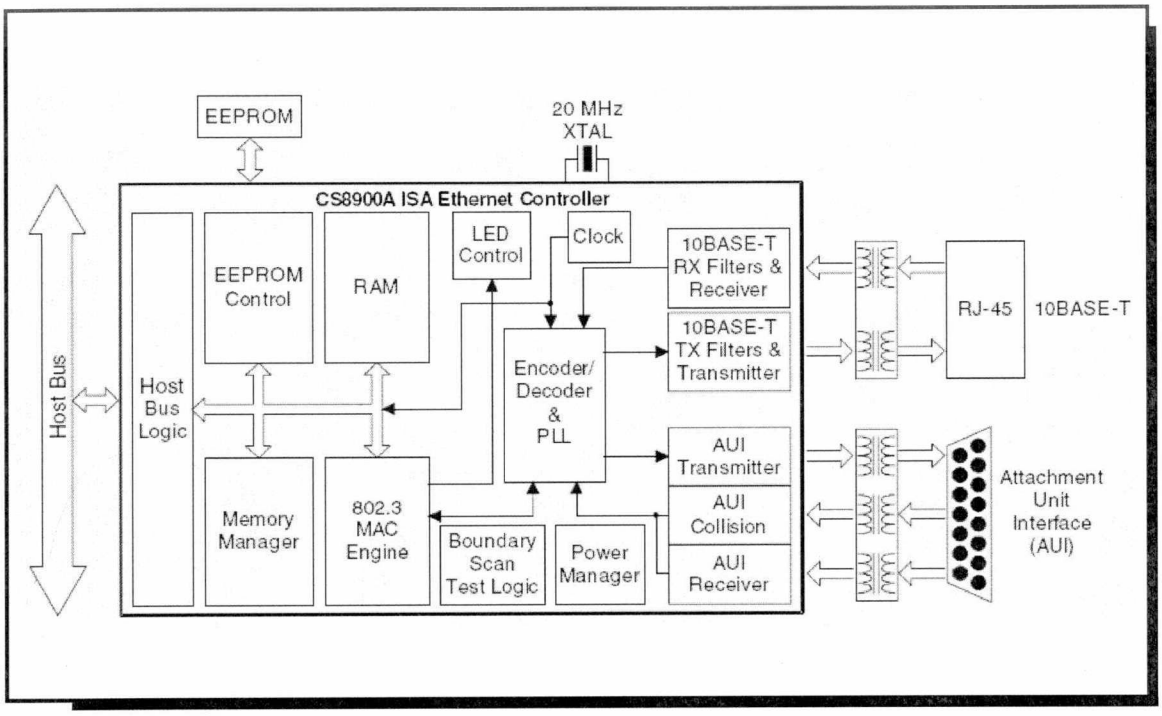


Figure 4-2: Functional block diagram of CS8900A [29]

The feature of CS8900A includes a highly integrated design, which eliminates the need for costly external components required by other Ethernet controllers, and broad range of performance and configuration options, which automatically adapts to changing network traffic and available system resources. The other important features of CS8900A include: [29]

- *Single-Chip IEEE 802.3 Ethernet Controller with Full Duplex Operation:* CS8900A is available in a 100-pin Low Profile Quad Flat Package (LQFP) ideally suitable for small, cost-sensitive Ethernet applications. With CS8900A, the

system engineer can design a complete Ethernet circuit that occupies less than 1.5 square inches of board space. The CS8900A is fully compliant with IEEE 802.3 Ethernet standard and provides full duplex operations on the network.

- *Efficient PacketPage Architecture Enabling Operations in I/O, Memory Space and as DMA slave:* CS8900A architecture is based on a unique, highly-efficient method of accessing internal registers and buffer memory known as PacketPage. CS8900A has 4 KB of integrated RAM used as a temporary storage for transmit and receive frames and for internal registers. This on-chip RAM buffers transmit and receive frames.
- *Support for wide physical interfaces:* The CS8900A includes an integrated 10Base-T trans-receiver compliant with Ethernet standards and includes all analog and digital circuitry needed to connect directly to a simple isolation transformer. CS8900A Attachment Unit Interface (AUI) provides a direct interface to external 10Base-2, 10Base-5, and 10Base-FL Ethernet trans-receivers.
- *LED drivers for link status and LAN connectivity:* CS8900A has three input pins that are used to control the LEDs or external logic. One of the LEDs is called LANLED; it goes low when the CS8900A transmits or receives a frame, or when it detects a collision. LED-2, called LINKLED, goes low whenever the CS8900A receives a valid 10Base-T pulse. LED-3, called BSTATUS, goes low whenever a transfer of received frame is detected over the ISA bus.
- *Standby and suspend sleep modes:* CS8900A supports three low power modes for power sensitive applications. The three modes include hardware standby, hardware suspend, and software suspend. Hardware standby helps PCs conserve power and the PC is temporarily disconnected from the 10Base-T cable. Hardware suspend consumes least amount of current of all the three modes. In this mode all the internal circuits are turned off and CS8900A is electronically isolated from the system. Software suspend mode is used to conserve power in applications like adapter cards that do not have power management circuitry.

- *Direct Industrial Standard Architecture (ISA) bus interface:* the CS8900A supports a direct ISA bus running at clock rates of 8 to 11 MHz. the CS8900A is optimized for 16-bit transfer, operating in either memory space, I/O space, or as a DMA.

4.2.3 Circuit Description

The MSP430F149 with 60 KB of flash, 2 KB RAM and 6 general purpose I/O ports is a suitable platform for a web server. The CS8900A, a low-cost Ethernet LAN controller, with its highly integrated design, reduces the amount and cost of external components. Most Ethernet controllers are designed with a PCI interface, but the CS8900A bus interface is simple enough to be connected to the microcontroller directly. The availability of CS8900A in a 3 Volt version adds another advantage to interfacing this device with the MSP430F149. The CS8900A is operated in I/O mode with 8-bit width data bus. The CS8900A 8-bit data bus is connected to I/O port 5 of MSP430F149. In CS8900A, a 4-bit address bus is used to access eight 16-bit I/O ports. Two control signals, IOW and IOR, are used to indicate the read or write access in progress. Figure 4-3 shows the hardware block diagram of MSP430F149 and CS8900A.

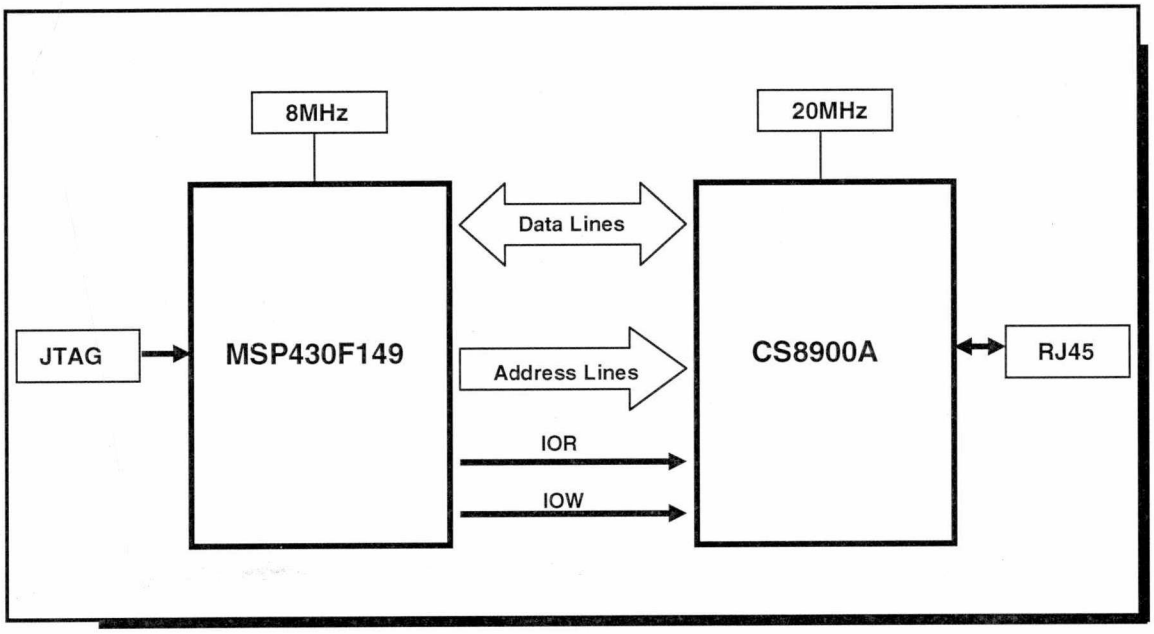


Figure 4-3: The hardware block diagram of MSP430F149 and CS8900A [6]

4.3 Software Description

Most embedded system engineers assume that developing a TCP/IP implementation is a highly time consuming process because they expect it to be a huge protocol. In reality, HTTP is a relatively straightforward protocol and can be implemented easily. The large number of embedded HTTP servers on the market bears the testimony to it [10]. This section describes briefly the TCP/IP stack implemented by Texas Instruments [6] using MSP430 and CS8900A, followed by implementation capabilities and limitations. This section later discusses the modifications made to develop the implemented web server into a fully-functional HTTP server. The implementation is described in two modules: Ethernet module and TCP/IP module. The Ethernet module describes the hardware driver for using the CS8900A LAN controller. This module explains configuring CS8900A, reading and writing into its registers, sending and receiving Ethernet frames.

4.3.1 Ethernet Module

The Ethernet module main tasks include Ethernet packet transmission and reception. The module has to be initialized and must be configured at power-up or at every reset for packet transmission and reception. The Ethernet software initializes the microcontroller input-output port pins, performs software reset of LAN controller, and sets up the LAN controller MAC interface. The CS8900A is initialized by entering configuration parameters into its internal configuration and control registers. The configuration values include the Ethernet hardware address (MAC address), media interface to be used, and the type of Ethernet frames that are accepted (Individually Addressed, Broadcast frames). The Ethernet hardware address/MAC address is an important network configuration that should be defined to the network interface. The MAC address is a 48-bit network interface identification address and should be a unique identifier on the local network. The six symbolic constants in the program, MYMAC_1 to MYMAC_6 defines the MAC address of the Ethernet module. The configuration values can either be sent by the host through the ISA bus or loaded automatically from an external EEPROM.

Once the Ethernet module is initialized, it can transmit and receive Ethernet packets. For CS8900A to transmit the packet, the host, MSP430 moves the Ethernet frames into the CS8900A buffer memory and the Ethernet module converts the frame into Ethernet packet and transmits it over the network. Figure 4-4 is a flowchart of the Ethernet module's transmit and receive operations. The MSP430 initiates the Ethernet frames transmission to the CS8900A by issuing a transmit command and then waits for the acknowledgement that the required buffer space is available. When the CS8900A is ready to accept the frame, the host writes the Ethernet frame into the CS8900A's internal memory. The CS8900A then converts the frame into the Ethernet packet by adding the preamble, start of frame, destination address, source address, length/type and a frame check sequence at the end. The data field must be between the lengths of 46 and 1500 bytes. If the data packet is less than 46 bytes, the field must include padding bytes to increase the packet size to 46 bytes. The data packet is followed by a 32-bit frame check sequence.

During the packet reception, the CS8900A receives the packet through the analog front, and the Manchester decoder decodes the Manchester encoded stream into the Non-Return to Zero (NRZ) data bits. The preamble and start of frame delimiter are stripped off, and if the frames destination matches the device's physical address the frame is stored in its internal memory. The received frame is then transferred to the MSP430 across the ISA bus.

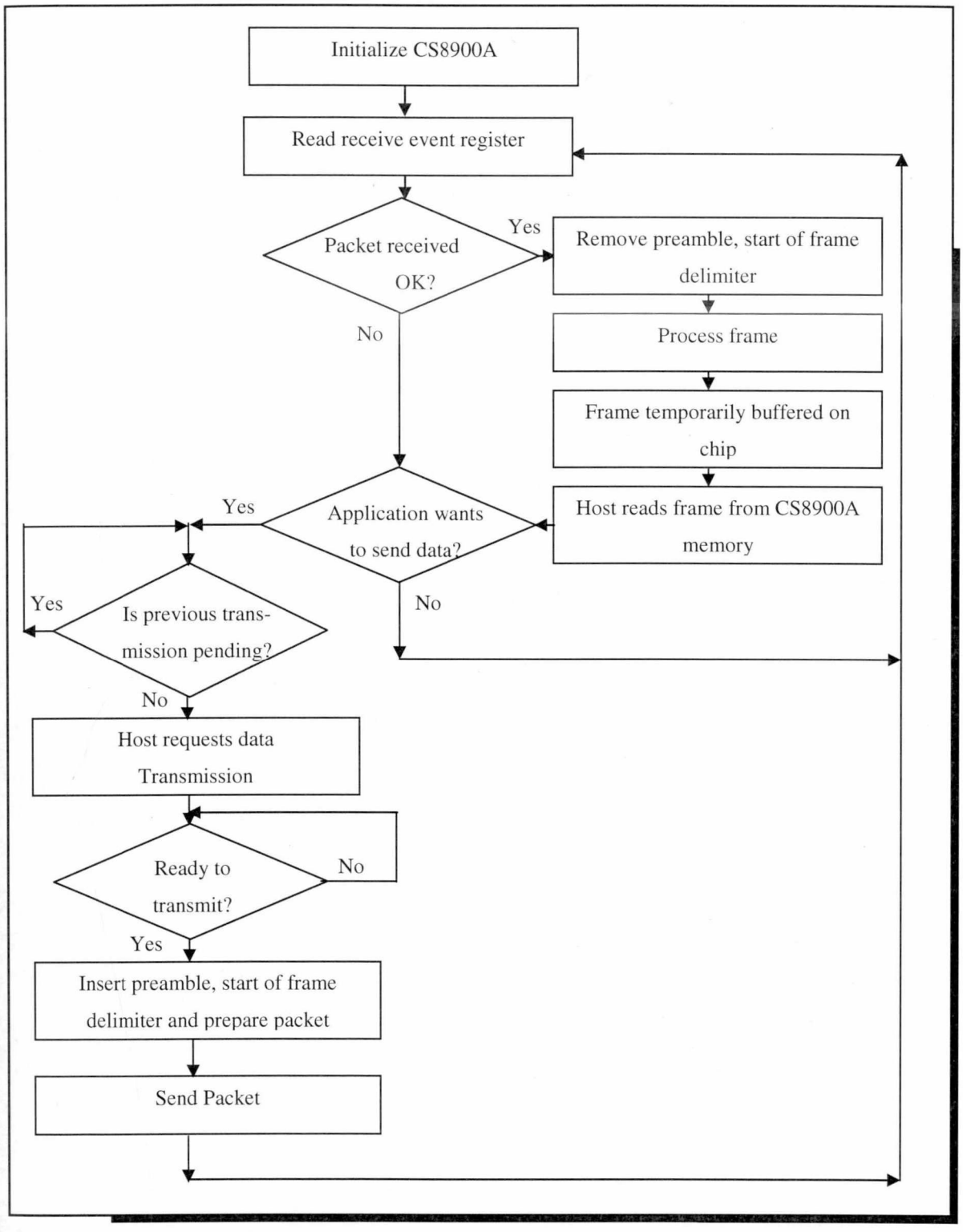


Figure 4-4: The flowchart of the Ethernet module transmit and receive operations

4.3.2 TCP/IP Module

The TCP/IP module implements all the necessary protocols for transferring data over the Ethernet network. The software module interfaces with the Ethernet module and uses its services to send and receive data and provide application programming interface (API) to the application layer.

IP transmits data between two individual computers on the Internet and TCP transmits data between two actual applications running on these computers. The IP address is the logical address or software address assigned to a computer on the network. Internet Protocol uses the IP address for data transfers between computers and TCP uses port number as its address. TCP uses IP for transferring data over the Internet. Since IP is a datagram-oriented (connectionless) service, there is not much to do for cases in which the IP datagram is not delivered. Hence, TCP has to take care of such cases. TCP is a connection oriented protocol; it establishes a connection between two applications for a reliable data transfer. The transferred data are numbered; the missing numbered data, either lost or damaged, are requested again. The transferred data is ensured by a checksum.

Together, TCP and IP handle the events that relate to transmission and reception of frames, opening and closing a TCP connection, timer control for retransmission of lost frames, network error control, etc. The MSP430 TCP/IP module continuously monitors the CS8900A receive control register. When a frame is received, it is de-multiplexed and after computing the necessary action the response is sent back to the Ethernet module to deliver the packet to the client. Figure 4-5 shows the flowchart of the network handling functions.

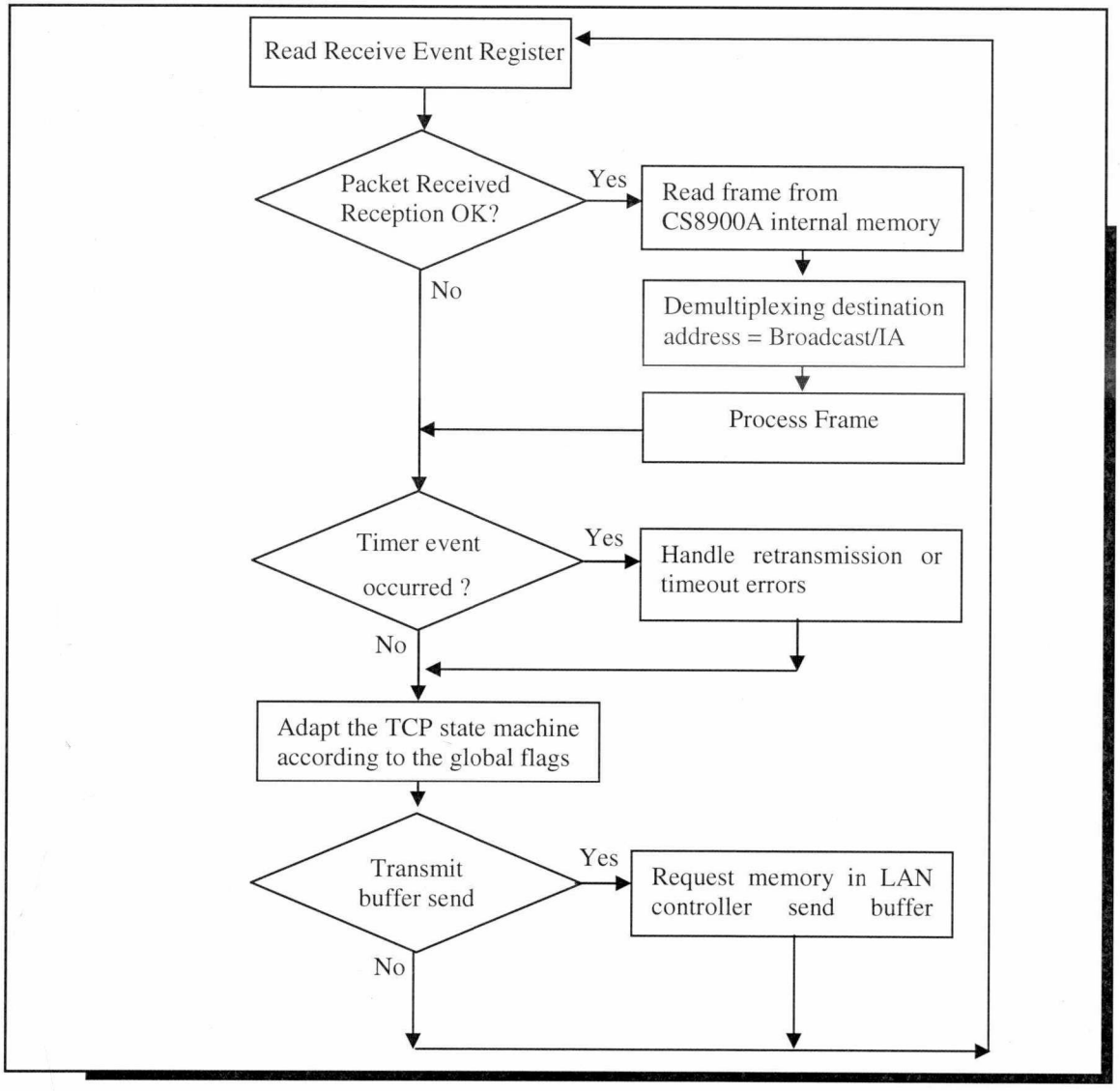


Figure 4-5: Flowchart of network handling functions

The process of de-multiplexing and parsing the received frame involves checking the destination address of the received frame, to determine whether the packet is individually addressed to the module or is a broadcast frame. If the frame is individually addressed to the module the source MAC address is copied and the Ethernet frame is processed to check the frame type. The frame can be an IP frame, or an ARP frame. If the frame is an ARP frame, the frame is checked to determine if it is an ARP answer for previously sent ARP request. If the received frame is an IP frame, the protocol type is

checked further to determine if it is an ICMP frame or a TCP frame. If the frame is an ICMP echo request the MSP430 TCP/IP stack generates an ICMP echo reply; other ICMP messages are ignored and discarded. If the frame is a TCP segment the frame is checked to determine if it belongs to the current active TCP session, or if it is a request for a new TCP connection (SYN flag in the TCP segment indicates a new TCP connection) and accordingly, the TCP state machine is updated and the response is processed. Figure 4-6 shows the de-multiplexing chart of a received frame. Appendix A includes all important protocol frame formats which are used during the implementation of TCP/IP.

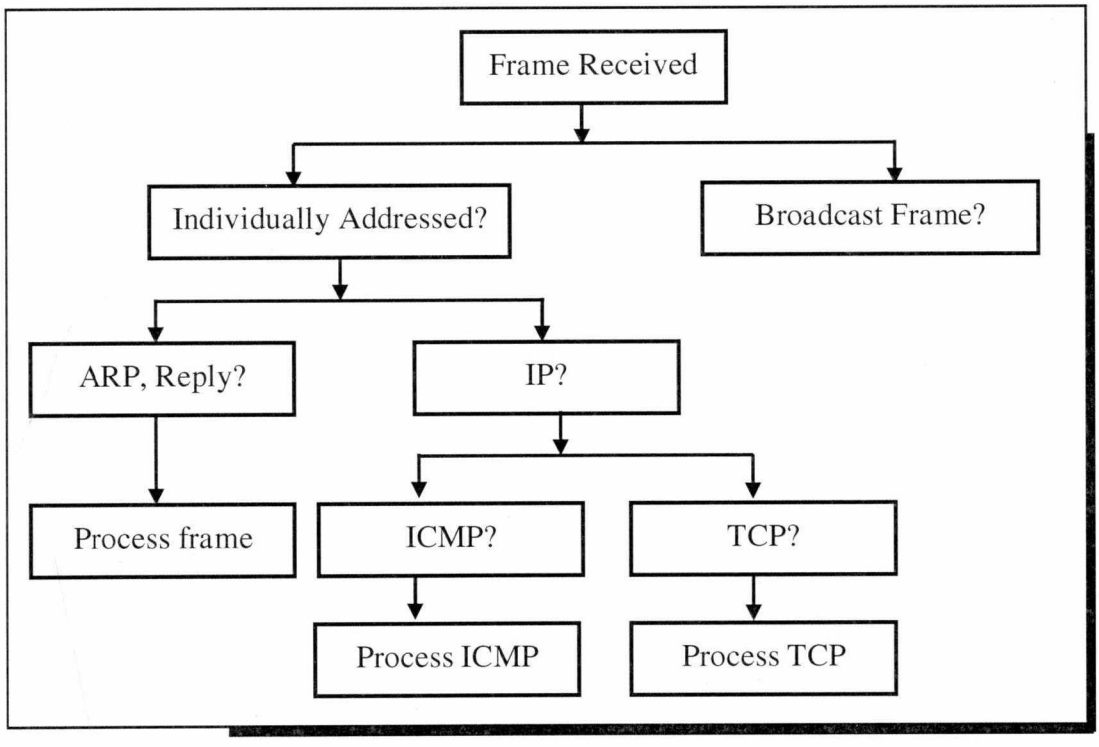


Figure 4-6: De-multiplexing of received frames [6]

Any specified target application on the Internet can be addressed with an IP address, a port number, and the protocol used. The IP address helps to identify a computer on the Internet and the operating system uses the target port number to recognize the application it should deliver the TCP segment. The source and destination

connections are specified with the port numbers. The TCP header reserves 2 bytes, each for source and destination port number. The 2 bytes can address ports from 0 to 65535 (2^{16} - 1). Port 80 is used by World Wide Web HTTP.

4.3.2.1 Opening and Closing a TCP Connection

A connection can be opened either in active mode or passive mode. The passive mode defines the TCP/IP stack to detect the incoming connection on a specified local TCP port. In active mode, the local and remote port numbers, and remote IP address must be specified. The TCP/IP stack tries to determine the remote MAC address by sending the ARP request. After finding the MAC address to communicate, the server opens the connection by setting the SYN flag and specifying the Maximum Segment Size (MSS). The majority of embedded web servers passively open a port and starts listening to connection attempts from clients. This process causes a number of changes in the TCP information maintained by the TCP module. The transitions are described by the TCP state diagram shown in the Figure 4-7.

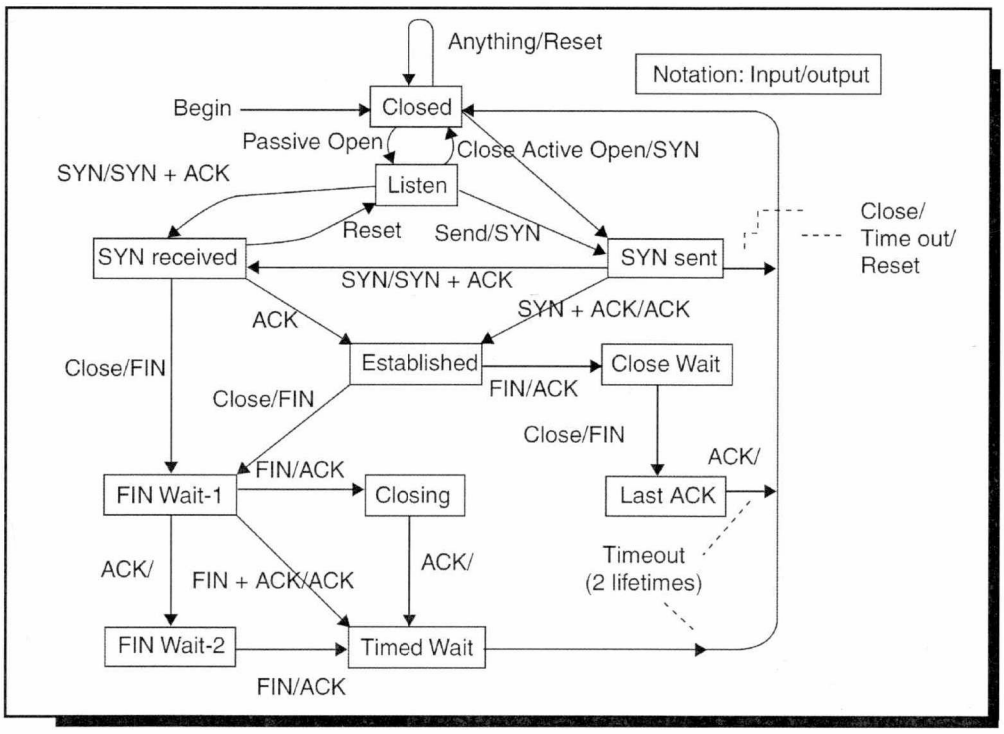


Figure 4-7: TCP state diagram [30]

The TCP establishes the connection between the client and the server using a 3-way handshake process. The client sends a packet with SYN = 1 and Initial Sequence Number (ISN) to the server. The server then replies with SYN-ACK (i.e. SYN = 1, ACK = 1), acknowledgement number = (ISN from client + 1) and Sequence number is set to the ISN of the server. The client sends a packet with ACK = 1, acknowledgement number = (ISN of server + 1), SYN = 0, sequence number = (previous sequence number + 1). Table 4-1 shows an example of a 3-way handshake connection establishment; the client and server TCP port numbers are taken as 1045 and 80 (HTTP) respectively and a random 32-bit ISN is used.

Once the connection is opened, the data transfer begins with the MSP430 copying the data into the CS8900A transmit buffer and the CS8900A takes care of the rest of the process to transfer the data on the physical network.

TCP uses a 4-way handshake to close the connection. The server sends a packet to the client with FIN = 1 (Finish) which indicates "no more data from server." The flag FIN is used to close the connection in the normal way, but the connection can also be closed using RST (reset) flag. The client receives FIN = 1, enters the CLOSE_WAIT state and sends a packet to the server with ACK = 1, acknowledging the FIN packet reception. The client sends another packet, but now with FIN = 1; the server confirms the close by acknowledging the finish, ACK = 1. No further packets are exchanged after this packet. With the last packet, the number of TCP packets sums up to four to close a TCP connection in a normal situation.

Table 4-1: An example showing the TCP flags and frame numbers while establishing a connection using 3-way handshake

SYN	ACK	Source Port	Destination Port	ISN	Acknowledgement number
1	0	1045	80	610,245,281	0
1	1	80	1045	12,357,945	610,245,282
0	1	1045	80	610,245,282	12,357,946

4.3.2.2 TCP Timers

TCP depends on highly controlled timing events. A connection establishing timer is started when the SYN is sent during the initial handshake process. The typical value of a connection establishing timer is around 75 seconds and when a time-out event occurs, the connection is aborted. FIN-WAIT (finish-wait) timer is started when a connection close is initiated by sending a FIN = 1 packet. The timer counts up to 10 minutes for the FIN-ACK; if a packet with FIN = 1 is received, the timer is cancelled. On the expiration of 10 minutes, the connection is dropped. TIME-WAIT timer is started when the connection enters the timed-wait state. The connection waits for 2 minutes to remove all the transit segments from the network. On the expiration of the wait time, the connection is terminated. MSP430 uses Timer-A to control all the timing events of the TCP connections. The 8 MHz crystal on the MSP430 is clocked down to 250 kHz and sources a 16-bit wide free running counter (TAR). The interrupt service routine is executed every 0.262 seconds by the microcontroller to control the TCP timers and is used to generate the ISN of the first TCP segment to the client. The timer controlled mechanism is also used to handle the retransmission of frames. Every time a frame is sent out to the network, the LastFrameSent register holds the last frame transmitted and in the case of acknowledgement time-out the last frame is transmitted again.

4.3.3 Application Programming Interface for WSN

The MSP430 TCP/IP stack is capable of listening to the HTTP requests from the clients and an application is developed to use the services of the TCP/IP stack and respond with the appropriate web content. The server provides the HTML page stored on the microcontroller flash memory. The module passively listens to the incoming HTTP requests. When a connection is established with a web client, the API handles the request and transmits the web page back to the client. After the data are sent successfully the TCP/IP closes the connection with the client and reopens the connection for the next request. This section describes the software implementation of MSP430 embedded web server. The implementation starts with the hardware initialization followed by initializing

the supporting flags, opening a TCP connection, waiting for the HTTP requests and processing the client's requests and replying with the appropriate web content.

The hardware initialization includes setting the input/output ports for the MSP430 and CS8900A and initializing the MSP430 oscillator, UART, and 12-bit ADC. The software initialization includes: setting the IP address, writing the initializing sequence to the LAN controller, resetting the flags, and starting the timer interrupt service routine. The initializing sequence for the LAN controller includes: the MAC address and the physical interface configuration. The local TCP port is set to 80 and a passive connection is open for the network requests. In other words, the server listens on "MYIP: TCP local port" for an incoming connection. When a connection is established, the server expects a simple GET request from the client. Since the server hosts only a single web page, the request is not evaluated and a web page which is stored in the microcontroller flash is served. The web page is stored as a C string constant and is made up of a stream of characters with HTML tags. The web page is not coded in any special way.

The HTTP server checks the transmit buffer and, as soon as the buffer is available, the pointer to the web page and the total number of bytes to be transferred is calculated and the web page transfer is started by sending the HTML header; the actual web page follows it. The header information holds the decoding information of the web page, and is used by the Internet browser to select the appropriate application to present the web page correctly to the user. In our case, the web page is a simple HTML page and the browser can display it directly without the help of any other application. Once the total web page is transmitted along with the header information, the TCP connection with the client is closed and a new connection is again opened for the next client to connect to the server.

This thesis extended the embedded web server application originally supplied by Texas Instruments [6] to allow remote monitoring and control of the wireless sensor network. The client is able to request the data from the wireless sensor network and to send the commands to the network through the embedded web server to control the

network operations. This required establishing a two-way communication between the server and the client, i.e. the server decodes the request from the client and acts accordingly. A set of application programs is developed to help the server handle multiple web pages. The server is programmed to issue requests to the network and retrieve the data from the network and send them back to the client. The web server establishes the communication with the sensor network by communicating with one of the wireless sensor nodes in the network through a USART interface.

The wireless sensor network is made up of many wireless sensor nodes monitoring the environmental changes and reporting the changes to the central node. Each wireless sensor node has a unique sensor ID while the number of sensor nodes depends on the application. The embedded web server is connected to a special node in the network which listens to the requests for network monitored parameters from the server and forwards the request to the central node. The central node responds with the data to the special node in its next allocated time slot. The special node then forwards the response to the server to display on the web page.

The wireless sensor network home page presents graphically rich, highly interactive controls to monitor the network. The home page acts as a central navigation point to the wireless sensor network controls. The web page includes the user options to select either monitoring a single wireless sensor node or all sensor nodes active in the network. The single sensor ID selection includes a drop down menu which shows all the active sensors in the network. The server requests the list of active sensor nodes in the network periodically and inserts the active sensor list in the web page on a fly when the page is sent to the host. The dynamic changing active sensor list is inserted into the web page to present the user with the list of all sensor nodes active at that moment.

The home page also includes the parameter selection to monitor a particular network managed parameter. Each parameter selected with either a single sensor or all sensors presents a different web page with the requested data. This shows the web server's capability of handling multiple web page requests depending on the user's choice of monitoring data. Figure 4-8 shows the snapshot of the wireless sensor network home

page with the temperature and battery voltage of an individual sensor as the network monitored parameters. Hosting and sending multiple web pages instead of a single default web page stored on the flash required implementing two-way communication between the server and the client.

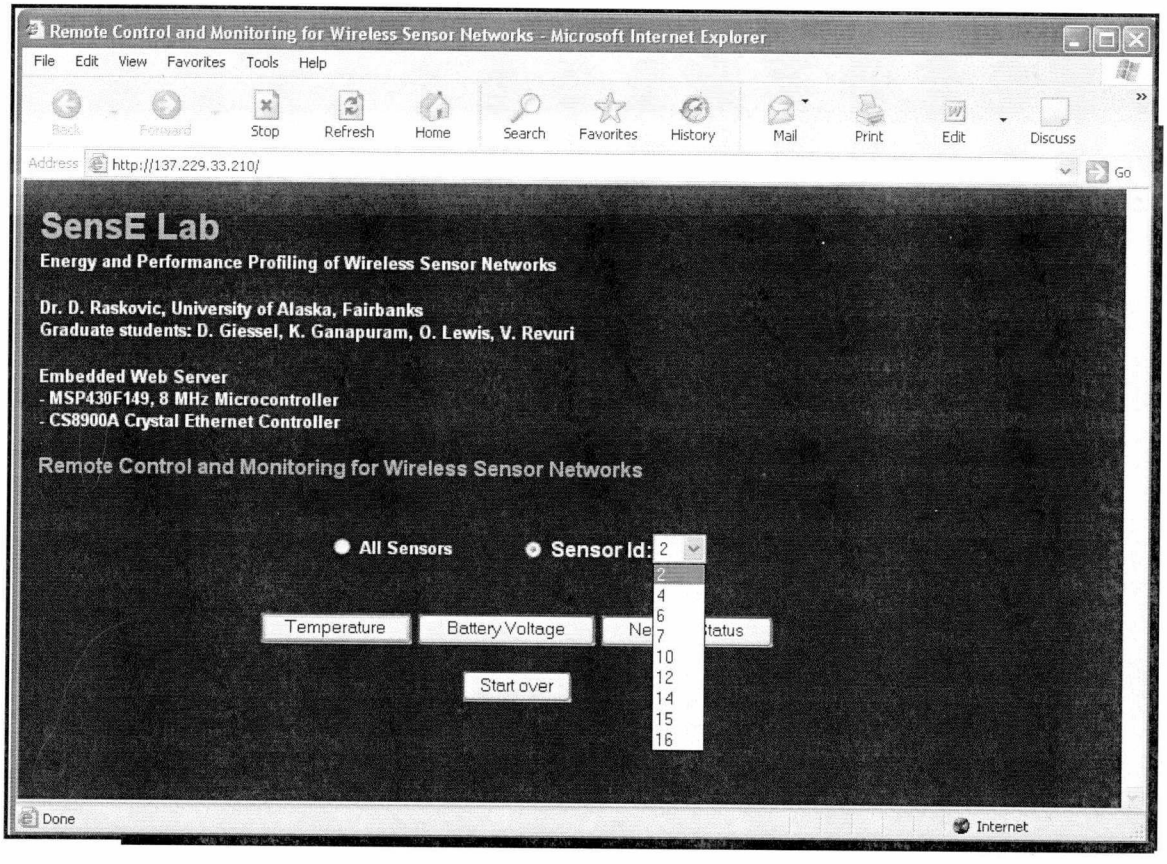


Figure 4-8: The snapshot of the wireless sensor network home page with the temperature and battery voltage as the network managed parameters

The server has to recognize client's requests and act accordingly. In order to implement the communication between the client and the server, we had to revisit the MSP430 TCP/IP implementation. The software implementation reveals that as soon as the connection is established, the server expects a simple GET request from the client. The server discards the received request without parsing the request and sends the default web page stored on the server flash memory. Modifications to the implementation fulfill the two-way communication. An application program is developed and added to the

software implementation to establish a two-way communication between the server and the client. After the modifications, the server is able to decode the received request and respond with the appropriate web content.

Similarly, modifications are made to the device web page to send the necessary request along with the device URL. As discussed earlier, GET is a simple request method of requesting documents from the server. A request sent using GET is in the form "GET document_path HTTP/version." Adding user input handling capabilities to a web page using HTML forms enables the client to send requests to the server. The request sent using GET and HTML forms is in the form "GET document_path/user_parameter _request HTTP/version." The user parameters entered in the web page are appended at the end of device URL and sent as a request to the server. The multiple parameters request is separated by '&' between the requested parameters. For example, the request sent to the MSP430 server for the temperature of a particular sensor ID is in the form "/feedback.msp?Choi=SNSID&SNSId=2&GetTemp=Temperature." Decoding the above request reveals that the user needs the temperature of sensor ID 2. After decoding the request, the server will send the appropriate web page with the requested parameter values. Inserting the dynamic content as in the case of parameter value of a specified sensor ID is discussed in the future sections. Figure 4-9 shows the snapshot of request and response of a single sensor temperature as discussed above.

Decoding the request from the web client allowed us to implement the web server hosting multiple web pages, which extends the data representing capabilities of the server. Similarly to storing the wireless sensor network home page in the microcontroller flash memory, other web pages are also included in the main program and stored on the microcontroller flash. Each web page is defined as a constant character array. During the transmission of the web page, the pointer to the character array is passed to the transmitting function and the total character array is enveloped in a packet and transmitted over the network. With multiple web pages, displayed data is better organized by presenting each network monitored parameter in a separate web page. The use of graphics made the web page easy to read and understand, and highly informative.

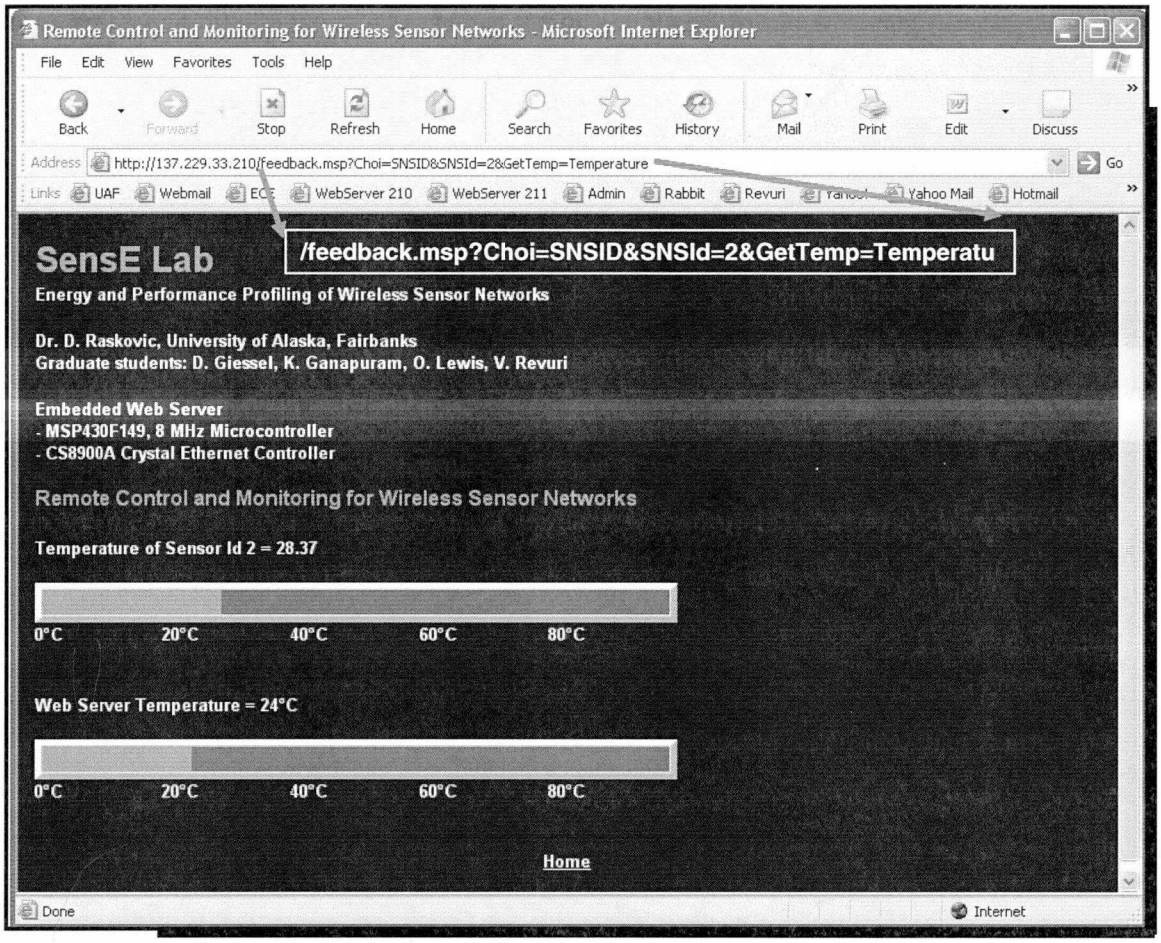


Figure 4-9: The snapshot of the request and response of single sensor temperature

With the increased use of graphics to present the data, the size of web page is increasing rapidly and a character array in a programming language could store only a limited number of characters. The number of characters in the array is limited to 4,000 (approximately), indicating the maximum web page size can be around 4 KB, and exceeding the limit would put us into problems. On the other hand, increasing the size of web page is important to present the parameter data of multiple sensors in a single web page. The parameter data of single sensor could be represented in a single web page easily, but the parameter information for multiple sensors was exceeding the limit of the character array. The general web server can host web pages with the size in the order of hundreds of kilobytes, but the MSP430 embedded web server is highly restricted in the case of web page size. Overcoming this limitation required developing an algorithm to

handle increased web page size by allowing a single web page to be defined by more than one string constant.

Wireless sensor network web pages are designed using Macromedia® Dreamweaver®, and the HTML code is copied into the string literal. If the web page is more than the single character array limit, then the web page is included into the server by defining it in more than one character array. Software implementation is modified by including many functions that can handle multiple strings for a single web page. When the web page defined by more than one string is called, the page header is transmitted first and the pointer to the first string definition of the page is passed to the transmitting function. Later, the next consecutive string is transmitted, and after successfully transmitting the last string, the TCP connection with the client is closed. Defining the web page in more than a single string constant does not affect the web server performance.

Unlike general web servers, the MSP430 web server does not have a file system; the entire web documents of the device have to be stored directly on the server's flash. The web page can either be written using a normal text editor or developed using web designing tools. MSP430 stores all the web pages directly as constant character array in the server's source code. Serving static web pages is easy for the embedded web server; the pointer to the character array is passed to the transmitting function in the web server's code and the whole character array is transformed into an Ethernet packet and transmitted over the network.

The dynamic web pages need some special attention to insert the changing data into the web pages. To serve dynamic web pages, the MSP430 uses the process of replacing special strings in the web page. The server stores simple web pages, and while designing the wireless sensor network web pages some special strings are embedded into the HTML code of the web pages. When a dynamic web page is requested, the server calls several functions in the server's main program to replace the special strings with the dynamic content before sending the web page to the client. These functions search the transmit buffer for the special strings. When such a string is found, it is replaced by the

current parameter measurement value and, after the search is done, the web page is sent to the client. Figure 4-10 highlights the request parameters and the special string for the dynamic data insertion in the wireless sensor network home page HTML code. The figure explains the features implemented in the web server principle to make the server a fully-functional HTTP server.

4.3.4 HTTP Server

Together, the TCP/IP module and the Ethernet module along with many modifications and added features implement a fully-functional web server. The server provides the home pages stored on the microcontroller flash memory.

The HTTP server waits for an incoming connection, transfers the web page, closes the connection, and waits for another client to connect. The web page content is dynamically updated using simple search and replace of special strings embedded into web pages. Along with the general text, the graphic rich web pages convey the information to the client. Figure 4-11 shows the MSP430-based web server operation using both the Ethernet module and TCP/IP module with all features included into the server's working principle. The flowchart summarizes the whole working principle of the MSP430 embedded web server.

Request sent to the server	Wireless Sensor Network home page HTML code embedded into the microcontroller	Dynamic Data inserted by the server
Request Method Type is GET /feedback.msp?Choi=SNSID&SNSId=2&GetTemp=Temperature	<HTML> <HEAD> <TITLE>Remote Control and Monitoring for Wireless Sensor Networks</TITLE> </HEAD> <BODY> <TABLE WIDTH="90%" BORDER="0"> <TR><TD> <P>SensE Lab</P> <P>Energy and Performance Profiling of Wireless Sensor Networks</P> <P>Dr. D. Raskovic, University of Alaska, Fairbanks Graduate students: D. Giessel, K. Ganapuram, O. Lewis, V. Revuri </P> <P>Embedded Web Server</P> <P>- MSP430F149, 8 MHz Microcontroller - CS8900A Crystal Ethernet Controller </P> <P>Remote Control and Monitoring for Wireless Sensor Networks</P> <FORM ACTION="/feedback.msp" METHOD="GET"> <TABLE WIDTH="40%" HEIGHT="63" BORDER="0" ALIGN="center"> <TR><TD WIDTH="50%"> <INPUT TYPE="radio" ID="NofSensors" NAME="Choi" VALUE="ALLSENS"/> <FONT> All Sensors </TD> <TD WIDTH="50%" HEIGHT="59"> <INPUT TYPE="radio" ID="NofSensors" NAME="Choice" VALUE="SNSID"/> <FONT>Sensor ID: </FONT> <SELECT ID="List" SIZE="1" NAME = "SNSId"> "OPT%" </SELECT> </TD></TR> </TABLE> <TABLE WIDTH="300" BORDER="0" ALIGN="center"> <TR> <TD WIDTH="33%"><INPUT TYPE="submit" NAME="GetTemp" VALUE="Temperature" /></TD> <TD WIDTH="33%"><INPUT TYPE="submit" NAME="GetBatt" VALUE="Battery Voltage" /></TD> <TD WIDTH="33%"><INPUT TYPE="submit" NAME="GetNetS" VALUE="Network Status" /></TD> </TR> </TABLE> </FORM> </TD></TR></TABLE> </BODY> </HTML>	**Special string "OPT%" is replaced with the dynamic content (Example as shown below) before sending the page to the client** <OPTION> 2 <OPTION> 4 <OPTION> 6 <OPTION> 7 <OPTION> 10 <OPTION> 12 <OPTION> 14 <OPTION> 15

Figure 4-10: Wireless sensor network home page HTML code

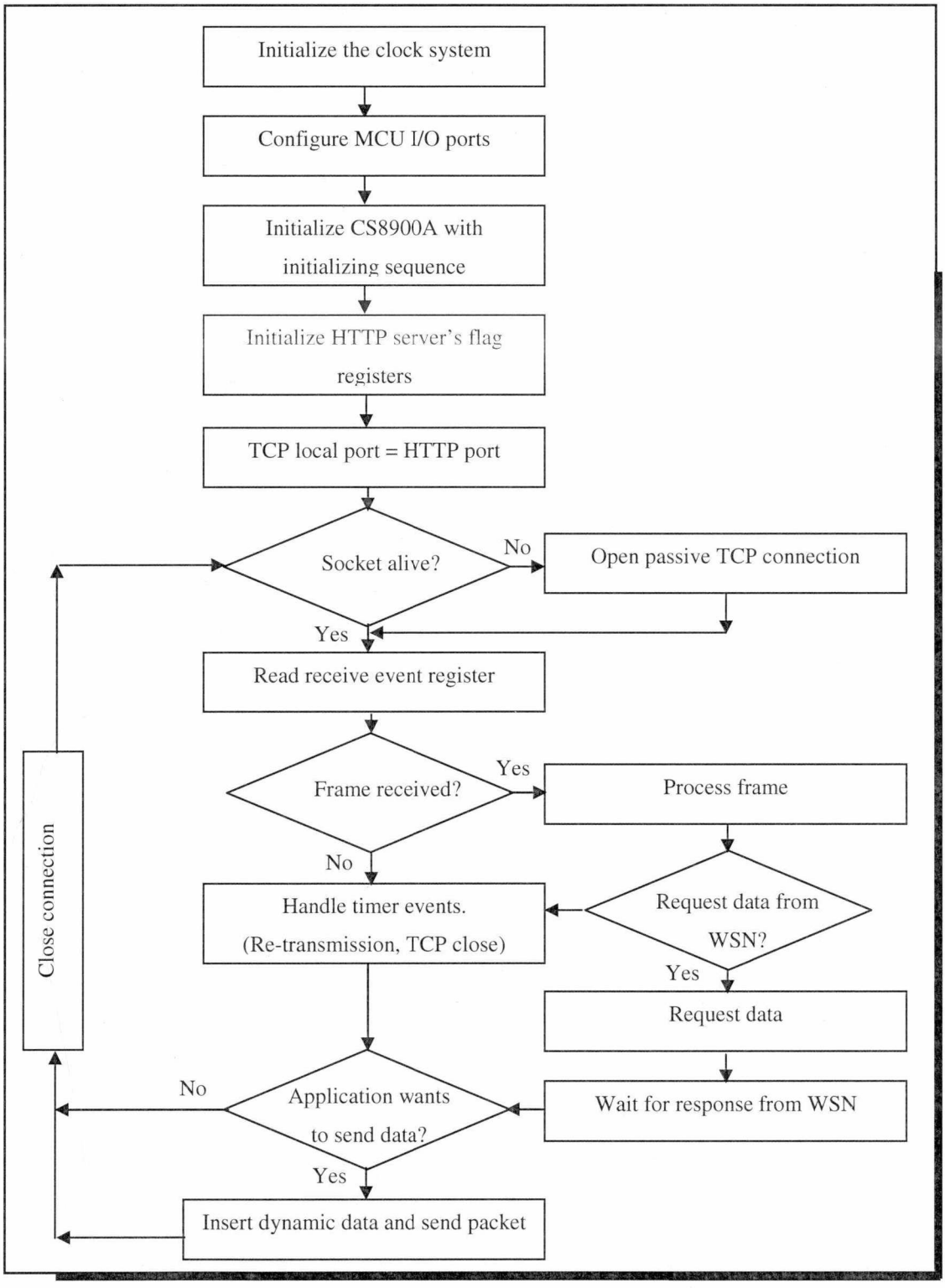


Figure 4-11: Flowchart summarizing the operation principle of the MSP430-based web server

5 Rabbit-based Web Server

5.1 Limitations of the MSP430-based Web Server

We used a MSP430F149-based board to implement a fully-functional embedded web server, an HTTP server implemented on a microcontroller with only 60 KB flash memory and 2 KB RAM. The MSP430 supports a 10Base-T (RJ-45) Ethernet connection. The TCP/IP stack is able to handle the protocols ARP, ICMP, TCP, and IP. It is optimized for low resource consumption. In spite of many compromises made during the software implementation of the HTTP server, the compatibility with other TCPs is very good. The MSP430 web server is integrated with the wireless sensor network enabling the client to issue commands to control the network and request data to monitor the network parameters. The server is capable of handling two-way communication. It's able to parse the request and act according to the request. The two-way communication opened the way to implement multiple web pages and the response to the client with the network monitored data is also successful. The dynamic network monitored data is embedded into the response to present the client with the current values "on the fly." The user management interface is highly interactive and provides the option to either monitor a single sensor node or all sensor nodes in the network.

In spite of the fact that the MSP430 embedded web server is fulfilling the objectives of the project, the TCP/IP stack and the web server have many restrictions. To ensure the low resource consumption, the TCP/IP stack supports only one active TCP session at a time. The server passively opens a connection and listens to the incoming HTTP requests. When a connection is established, the server processes the request and closes the connection with the client, reopening the connection for the next client to connect. The TCP/IP implementation does not reassemble the fragmented incoming IP frames; hence the HTTP requests are restricted to only single Ethernet packet. The TCP/IP implementation does not buffer the TCP segments which are delivered out of order, compute the checksum of incoming data, support the IP type-of-service (TOS), or support the security options. The web server also has some restrictions; the size of the web page should not exceed twice the buffer capacity of character array. Because the web

server has no virtual file system, pictures (gif or jpeg) cannot be used. The server supports only the GET request method, restricting the request data length to only 256 characters; the server does not support the POST request method.

With great restrictions on web page size, dynamic properties of the web page are tightly constrained. The memory on the web server is not of sufficient size to support many web pages. The memory size also places a restriction on data collection capabilities of the web server. The MSP430 is good at monitoring the wireless sensor network, but falls short in data collecting capabilities. The web server does not support any security options, making it highly susceptible to overloading the wireless sensor network with requests and jamming the wireless sensor network communication. All the above limitations and restrictions forced us to re-evaluate the design issues of the web server and adapt a microcontroller which can and provide better resources for the implementation of an embedded web server.

Rabbit semiconductor developed the RCM3750 RabbitCore™ module featuring Rabbit 3000® microcontroller running at 22.1 MHz, 10/100Base-T connectivity, 512 KB of flash and 512 KB of SRAM, 1 MB of serial flash, and a small footprint. Using the RCM3750's dual row IDC header, the module can be interfaced with a CMOS or 3.3 Volt compatible digital devices to create a web-enabled device. Figure 5-1 shows the top and bottom view of RCM3750. The RCM3750 receives regulated DC power supply from the web-enabled device motherboard on which it is mounted. The 33 parallel digital I/O shared with serial ports, power and other signals directly route to the motherboard. The RCM3750 is programmed with Rabbit Semiconductor's Dynamic C® programming language over a standard PC serial port. It can also be programmed through a USB port using any standard RS-232/USB converter. Together, the combination of Rabbit 3000 microcontroller, RCM3750, and Dynamic C makes a perfect solution for Ethernet/Internet based data collection and control for a distributed wireless sensor network.

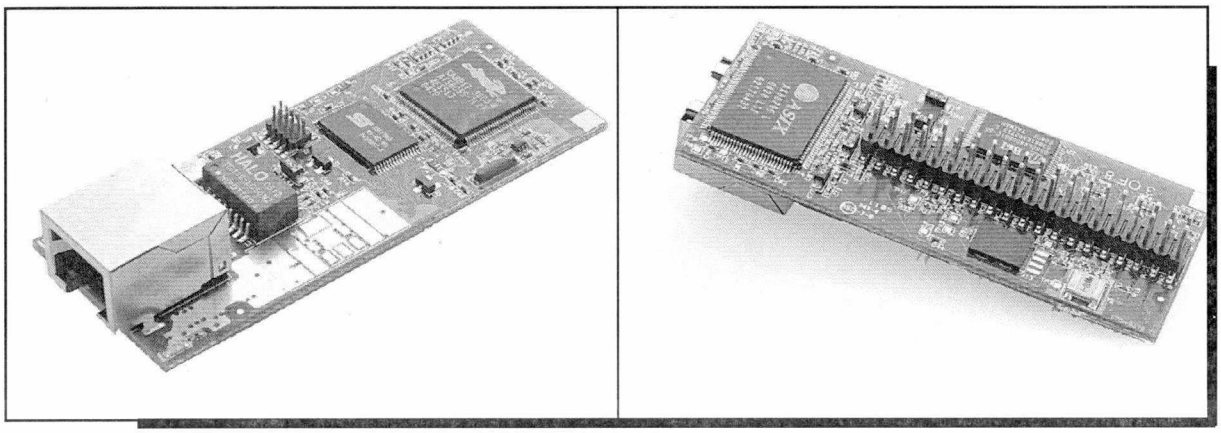


Figure 5-1: Top and bottom view of RCM3750 [31]

5.2 Hardware Description

The following subsections discuss the key features and design issues of Rabbit 3000 microcontroller, RCM3750, and Dynamic C.

5.2.1 Rabbit 3000®

The Rabbit 3000 is a high performance, low EMI microprocessor designed specifically for embedded control, communication, and Ethernet connectivity [32]. The 128-pin LQFP package, low operating voltages between 1.8 and 3.6 Volts, and clock speeds up to 44 MHz, make it a low cost, industrial and commercial friendly microcontroller. The key features of Rabbit 3000 microcontroller include: [26]

- 8-bit data bus
- 20-bit address bus
- High frequency clock and 32,768 Hz clock
- Built-in clock doubler
- 4-levels of interrupt priority
- 56 I/O signals (shared with serial ports and connecting other peripherals)
- Six UARTs
- Built-in watch dog timer
- 65mA @ 30 MHz, Vcc = 3.3 Volts

Rabbit 3000's 20-bit address, 8-bit data bus, three chip select lines, two output-enable lines, and two write enable lines helps it to directly interface with up to 6 flash/SRAM devices. Up to 1 MB of memory can be directly accessed using the standard Dynamic C developing environment allowing C programs with more than 50,000 lines of code; up to 6 MB of memory can be interfaced using the additional software development kits.

Figure 5-2 shows the Rabbit 3000 hardware architecture. The Rabbit 3000 CPU uses separate buses for external and internal data transfers. The CPU registers act as source and destination for all external bus transfers. The internal buses include an 8-bit data bus and 8-bit address bus. The Rabbit 3000 has seven 8-bit digital I/O ports. These 56 I/O pins share functions with other on-chip peripherals.

Even though the Rabbit 3000 is designed to operate between 1.8 to 3.6 Volts, all its I/O ports can be interfaced to 5 Volt logics. Rabbit 3000 has six serial ports and all six ports can be used as simple three wire asynchronous serial ports with maximum bit rate of the system clock divided by 8. Rabbit 3000 has two timer modules, Timer-A and Timer-B. The Rabbit 3000 provides a clock doubler, which can double the slow speed external crystals.

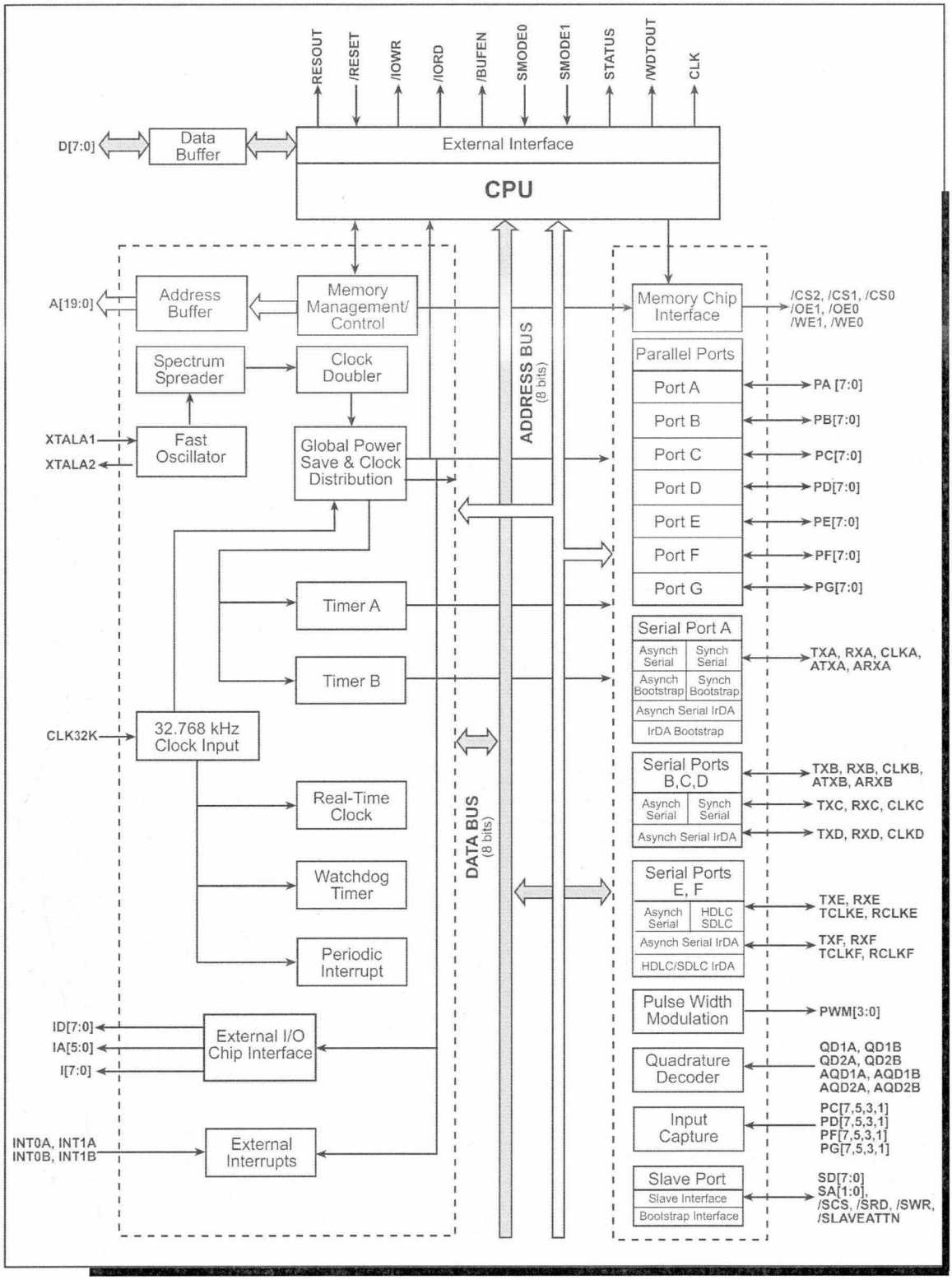


Figure 5-2: Rabbit 3000 block diagram [33]

5.2.2 ASIX – AX88796L

The AX88796L Ethernet controller from ASIX is high performance and highly integrated with the 10/100 Mbps MAC, PHY, and transceiver making it a 3-in-1 local CPU bus Ethernet controller. The AX88796L is a 128-pin LQFP low profile package which operates at 25 MHz with a 3.3 Volt supply. The AX88796L consumes very little power, typically under 100 mA, making it a good choice for power consumption sensitive products like PDA, embedded products, etc. The features of AX88796L include: [34]

- *Embedded 8K x 16-bit SRAM and EEPROM interface:* The AX88796L uses four buffer memory access types for its local and remote DMA read-write operations. The AX88796L supports EEPROM interface to store the device MAC address.
- *10/100Base operations:* The Ethernet chip supports both 10 Mbps and 100 Mbps data rates with both full-duplex and half-duplex operations based on the IEEE 802.3 and 802.3u LAN standards.
- *Supports 8- and 16-bit CPUs:* The AX88796L supports 8-bit and 16-bit local CPU interfaces including MCS-51 series, 80186 series, and MC68K series CPU.
- *Media-Independent Interface (MII):* The AX88796L provides an extra IEEE802.3u compliant MMI to support media like home LAN applications. It also provides an optional standard parallel port interface to support a printer. Figure 5-3 shows the block diagram of AX88796L with optional home LAN PHY and printer port.

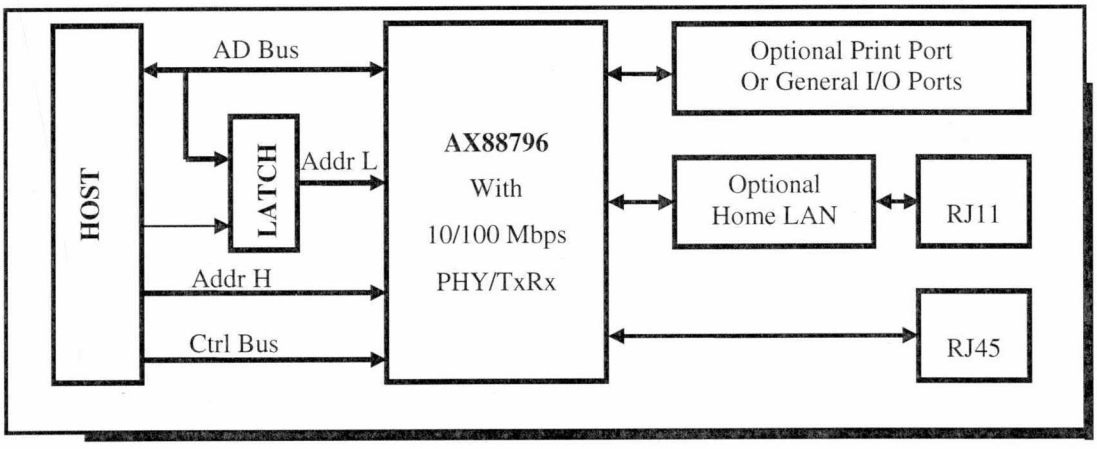


Figure 5-3: ASIX - AX88796L block diagram [34]

5.2.3 RCM3750

The RCM3750 is a compact embedded web server which incorporates the latest revision of the Rabbit 3000 microcontroller, flash memory, onboard serial flash, static RAM, and abundant digital I/O ports. The RCM3750 features an integrated 10/100Base-T Ethernet connection for the Internet based systems. The RCM3750 has a Rabbit 3000 microcontroller, static RAM, flash memory, two clocks (main oscillator and real time clock), and circuitry necessary for reset and battery management of Rabbit 3000's internal real time clock and static RAM. The RCM3750's dual row IDC header brings out the Rabbit 3000's I/O bus lines, parallel ports, and serial ports to interface with the web-enabling device. The RCM3750 receives regulated DC power from the web-enabling device on which it is mounted. Figure 5-4 shows the block diagram of RCM3750 RabbitCore.

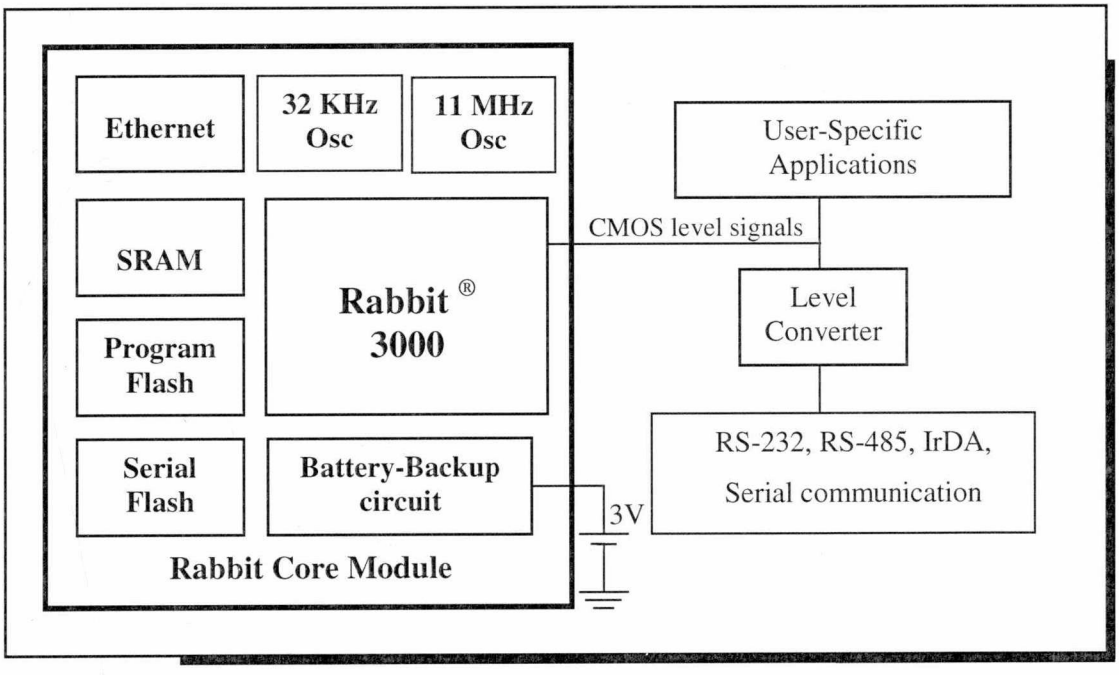


Figure 5-4: RCM3750 subsystems [35]

The RCM3750 RabbitCore specifications and features include: [35]

- *Microcontroller:* The RCM3750 has a Rabbit 3000 microcontroller operating at 22.1MHz which handles all the core functionalities and operations of the

RCM3750 including memory management, external interfacing with other peripherals, Ethernet connectivity, etc.

- *Digital I/O:* The RCM3750 provides 33 parallel 5 Volt tolerant I/O lines: 31 configurable for I/O and 2 fixed output pins; and an external reset pin is included as an additional input.
- *Timers:* The RCM3750 has ten 8-bit timers (six timers can be cascaded together and three are reserved for internal peripherals) and one 10-bit timer with two match registers. These timers help the RCM3750 to keep track of all TCP/IP timing requirements.
- *Ethernet connectivity:* The RCM3750 provide the 10/100Base-T RJ-45 Ethernet port, which supports both half- and full-duplex operations. The RCM3750 also supports a wireless Internet interface.
- *Memory management:* The RCM3750 includes 512 KB flash memory manufactured by Silicon Storage Technology (SST) and 512 KB static RAM manufactured by Brilliance (BSI).
- *Serial flash:* The RCM3750 includes 1 MB serial flash memory, making it a great choice for data collecting applications.
- *Serial ports:* The RCM3750 has four shared high speed, 3.3 V, CMOS-compatible serial ports. The serial ports are designated as C,D,E and F. Besides operating the serial ports in asynchronous mode, the serial ports C and D can be operated in clocked serial mode; and serial ports E and F can be operated in HDLC serial mode.
- *Board size and connectors:* The RCM3750 has a small footprint with the dimensions 2.95 x 1.20 x 0.89 inches (75 x 30 x 23 mm).

5.2.4 RCM3700 Prototyping Board

The RCM3750 is available both as an individual core module and as a part of complete Ethernet development kit. The development kit includes the RCM3750 RabbitCore, development board with prototyping area, Dynamic C development system,

and a serial cable used for programming and debugging. The RCM3700 prototyping board has power supply connections and provides basic I/O peripherals (RS-232, RS-485, ADC, LEDs, and switches) and a prototyping area for more advanced hardware development. Figure 5-5 shows the RCM3700 prototyping board.

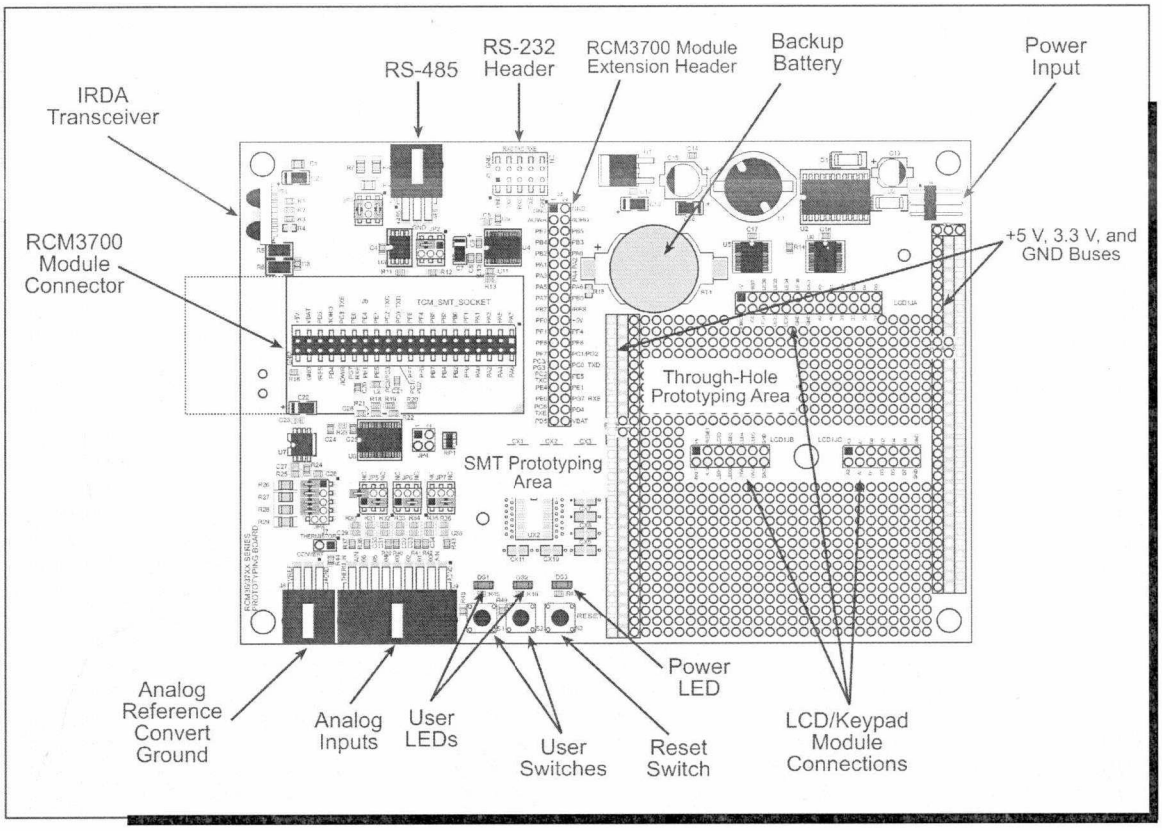


Figure 5-5: RCM3700 prototyping board [35]

The features of prototyping board include:

- *Power connections:* A 3-pin header provides the power supply to the board. The 3-pin header is symmetrical and can be connected to the power input either way. The prototyping power supply ranges from 7.5 Volts to 30 Volts DC at 500 mA. The onboard power regulator provides a stable 3.3 Volts to the RCM3750 RabbitCore module.
- *LEDs:* The power LED indicates the power ON to the complete board. The other two LEDs connected to port F pin-6 and port F pin-7 of the RCM3750 module can be driven as debugging output LEDs.

- *I/O switches:* The reset switch is directly connected to RCM3750's RESET pin; when pressed, it forces a hardware reset. The other two switches are connected to port F pin-4 and port B pin-7 of the RCM3750 module, and can be read as inputs to debug RCM3750 operations.
- *Prototyping area:* The prototyping area provides room for installing through-hole components used for advanced hardware development.
- *LCD/Keypad module:* The prototyping board provides room for plugging optional LCD and keypad modules.
- *Serial ports:* Three 3-wire serial ports or one 5-wire and one 3-wire RS-232 serial ports are available at header J2. Jumper JP2 controls the serial port E to work either in 3-wire or as an RS-485 serial port.
- *IrDA:* The prototyping board includes an infrared transceiver capable of handling links up to 1.5 meters.

5.3 Software Description

This section discusses the Rabbit programming environment, the TCP/IP implementation of the Rabbit web server, and the application programming interface developed to web enable the wireless sensor network.

5.3.1 Dynamic C Programming Language

Many companies offer controllers, but few offer hardware and tightly coupled software libraries. This makes the software engineer spend a great deal of time to develop even a small, rudimentary application for the controller. This fact is overlooked by many engineers, but a Rabbit microcontroller comes with complete built-in software libraries. All Rabbit microcontrollers are programmed using Rabbit Semiconductor's Dynamic C. The Dynamic C environment includes an editor, crosscompiler, downloader, and in-circuit debugger. Using the crosscompiler, the software developer can write and compile the program for his/her board on Window's platform, and download the executable code to the microcontroller core. The following is the summary of features of the developing environment:

- Dynamic C includes an integrated development environment, eliminating the need for separate editors, compilers, assemblers or linkers.
- Dynamic C has an extensive library of drivers making the coding easy and decreasing the application development time.
- Dynamic C uses serial communication to download code into a target device, eliminating a need for CPU or ROM emulator.
- Dynamic C also supports assembly language programming. Assembly language can be integrated into normal code. C and assembly language can be mixed together.

Dynamic C is not an ANSI C. Rabbit Semiconductor developed Dynamic C, a programming language (system development) to meet the unique performance demands of embedded controllers. Dynamic C provides extensions to the C language which support real-world embedded system development.

5.3.2 TCP/IP Module

After discussing the software development environment and hardware from Rabbit Semiconductor for web-enabling the wireless sensor network, this section presents the TCP/IP implementation of the Rabbit web server. The TCP/IP standard defines the rules of communication on TCP/IP networks. TCP/IP implementation is the software component which performs the function of enabling the computer to become involved in the TCP/IP network. The details of the TCP/IP standard are discussed in Chapter 2, Ethernet basics, and Chapter 4, MSP430-based Embedded Web Server. The software component is the vendor's implementation of TCP/IP, and this section describes the TCP/IP implementation and network configurations of a Rabbit web server.

The TCP/IP stack configuration includes the selection of an interface and the necessary software functionalities for a particular application. The necessary software functionalities include selection of TCP, UDP, or both protocols, and selection of DHCP, depending on the application. Including only the functionalities needed for a particular application frees up room for the application programming interface. Dynamic C has a feature to remove unwanted protocols in the TCP/IP implementation via conditional

compilation. This is accomplished by defining the configuration macros in the server program. The interface configured for web-enabling the wireless sensor network is a standard Ethernet connection defined by a single macro USE_ETHERNET, which includes the Ethernet drivers and initializes the Ethernet module AX88796L.

To run the TCP/IP stack, the server has to configure the network controls. The server needs to have its unique IP address identifying the server on the network. Interfaces defined on Ethernet must also have the net mask assigned. Together, the IP address and the net mask describe the subnet, which can be directly addressed by the host. Apart from the IP address and net mask, other important network configurations include the gateway host which forwards messages between the local host and the outside world. The Dynamic C TCP/IP stack obtains the network configuration information from any of the following sources: a predefined configuration defined in TCP configuration library, macro definitions for static configuration, bootstrap network protocols such as DHCP for dynamic configuration, etc.

The TCP/IP stack initialization starts by calling the subsystem initialization for ARP, TCP, DNS (if applicable), resets the Ethernet hardware and clears out the packet receive buffer pool, and clears router and other server tables. Interfaces are initialized using the settings defined in the TCP configuration library file. If DHCP is specified, the server issues a request to a DHCP server for a dynamic logical address (IP address) on the network. Once the initialization is successful, the server opens a passive connection and waits for the client to connect to the server through a normal three-way handshake. The Rabbit web server is capable of listening to HTTP requests from the client and responding with the appropriate web content. It includes all the capabilities of the MSP430 web server; hence the whole discussion is not presented again in this section. The Rabbit server handles the request to monitor a network parameter of either a single wireless sensor node or all active sensor nodes in the network. The client has an option to browse through any parameter which is monitored by the wireless sensor network. An extra-feature included in the Rabbit web server that could not be implemented using the

MSP430 board is the increased dynamic nature of the graphical representation of all sensor nodes' parameter values.

The MSP430 web server presents the parameter values of all the sensor nodes, including the sensor nodes which are not currently active in the network. We improved the dynamic capabilities of the web page using the Rabbit web server that displays the parameter values of only those sensor nodes that are currently active in the wireless sensor network. This helps represent the network data in a more organized way, without including redundant values (N/A). The difference will be clearly visible in the case of a sensor network with the maximum node capacity of 256, of which say, only 64 sensor nodes are currently active. The rest of the section discusses the extra-features and the capabilities of the web-enabled wireless sensor network. Before describing the server application, it is helpful to know how server application is organized. Figure 5-6 shows the web architecture of the Rabbit web server, highlighting all the relevant components of the web-enabled wireless sensor network. The lines between the components represent the communication between the components and the arrows represent who is accessing whom.

5.3.2.1 Application Block

The top block of the web server architecture is the application block. The application block includes: compile time initialization block, runtime initialization block, the main loop, application specific I/O block, and CGI functions block. The five sub-blocks describe the major part of the application programming interface of the web server. The sub-division of the application block handles specified tasks in the server operation. The following paragraphs describe the sub-division tasks in detail.

- *Compile-time initialization:* This sub-division of the application block initializes the static (constant) data structures and tables, selects the default network configuration and includes static resources to load into the server application using #ximport and #zimport directives.

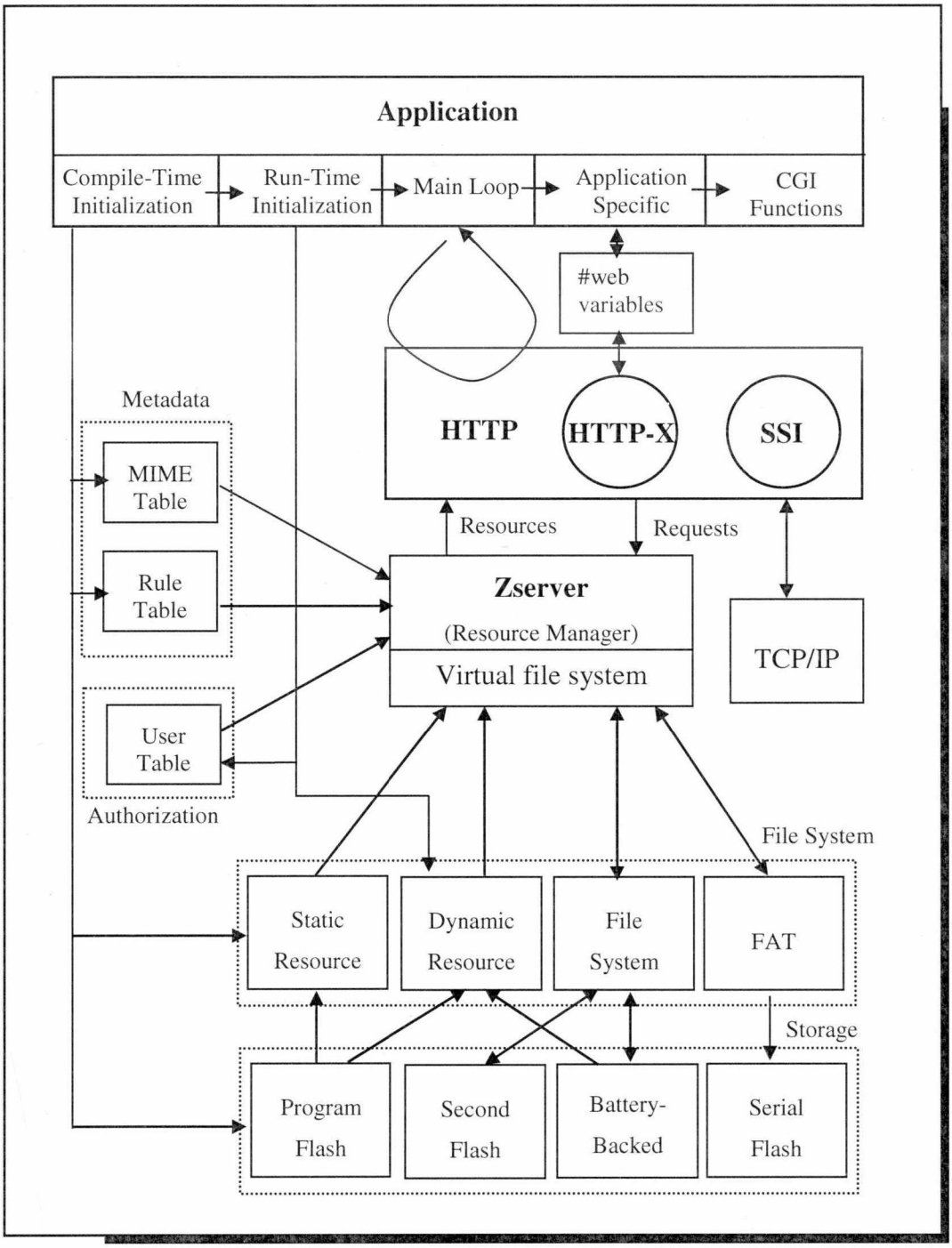


Figure 5-6: Web architecture of Rabbit web server [36]

The arrows starting from the "compile time initialization" sub-block indicate the tables that are setup at compile time.

- *MIME mapping table:* Multi-purpose Internet Mail Extension (MIME) is a standard which defines the types of files attached to the server application. It is used in the computer program which needs to communicate with another program about what kind of file is being sent. This mandatory table describes to the browser how the content is supposed to be displayed.
- *Rule table:* The rule table is only necessary when the server is using a file system. The rule table sets the access permissions to all the files in the file system.
- *Static resource table:* The static resource table defines the resources in Dynamic C. The application with static resources, or data that will never change, such as those making up an image, are defined in the static resource table.

Program flash: Program flash is the memory where the resource files are loaded using the #ximport directive. All the web pages of the web-enabled device are stored here, and the memory is limited to only few kilobytes.

Runtime Initialization: When the server is started, runtime initialization calls the specific library functions used by the application. This block describes all mandatory functions used to initialize the TCP/IP networking system. The optional available FAT file system is also initialized here for use by resource manager Zserver. Runtime initialization includes initializing the HTTP server to set the working of web server in HTTP mode. Various other functions including user table, dynamic resource table are initialized at run time.

Main Loop: The main loop is an endless routine that handles the HTTP functions to continuously stay on the network. This function handles all the frame types and protocols, like ARP, ICMP, and HTTP, to set the server running.

Application Specific and I/O: This is the actual application specific block, where all the functions related to the web-enabling device are defined. The application specific functions include: handling requests from the client, performing the requested operations, and responding with the appropriate web content. The application specific functions also include communication between the server and the web-enabling device. The application

interfaces with the HTTP server to establish the communication between the client and web-enabling device.

CGI functions: CGI functions refer to 'C' functions called by the HTTP server to generate the dynamic content embedded into the web pages. CGI functions are called upon a request from the client for the data. The CGI function generates the current device status, collects the requested data, and responds to the client with this dynamic content embedded into the web pages.

5.3.2.2 HTTP Block

HTTP block represents the HTTP server itself. The server is responsible for all the functions of the web server which include listening to requests from the clients and when connected to the client, analyzing the request, determining the source of the request for sending the response, determining the user authorization and access controls. When the user has the proper permissions the server transmits the requested resource back to the client. The following steps explain the above functions in detail:

- *Analyze the request:* When the client requests any resource from the server, the URL carries the request to the server and the server parses the request and locates the resource with the path specified by the URL. The path is specified as in the UNIX-style file system with the directory and the file name separated by a slash (/) character.
- *Obtain the user ID:* With the request for resource, the browser has the option of sending the user name and password of the client. If the browser does not provide user credentials, the user ID is assumed "anonymous" and the HTTP server consults the resource manager as to whether the requested client is authorized to get the resource. The resource manager in turn looks up the rule table and determines the group of which the user is a member. Depending on the group, the user is checked with the allotted permissions and returns the HTTP server a YES or a NO answer.
- *Return the resource:* After the user credentials are validated, the resource is transmitted back to the server. The resource can be in the form of an HTML file

or an image and it is read from the program memory or flash system. The resource can also be generated "on the fly" and transmitted back to the client. The resource can also call a CGI function, and the response includes the result from the CGI function.

The HTTP block is made up of sub-components which include RabbitWeb (HTTP-X) and SSI. RabbitWeb is an add-on module to Dynamic C. It acts as a web complier and connects the web variables (as in the case of special strings embedded into the web page) defined in the device web pages to the server and the application. The block named *#web variables* defines the variables which act as a means of communication between the application and the server. *#web variables* are just ordinary C constants, arrays, and structures. RabbitWeb allows the user to develop a web page in a special scripting language, making it easy to generate web pages and apply dynamic capabilities to the web page.

SSI, also a scripting language, is used to generate the web pages "on the fly." The SSI is similar to the RabbitWeb but has an extra control on CGI functions. The SSI can generate web pages and include the dynamic content by calling the CGI functions, but SSI programming can become fairly complicated.

5.3.2.3 Zserver Block

Zserver block is the resource manager and the central processing unit of the web architecture. It controls all the components, and every component uses the services of the resource manager to implement its tasks. Zserver is implemented both as a resource manager and a virtual file system. The virtual file system is basically a notational convenience for accessing all the resources using a uniform naming scheme [36]. The Zserver, as a resource manager, takes the responsibility of mapping the various file systems and resource types into a single unified application programming interface. It also provides services to various components and all the components report to the resource manager with their services.

5.4 Application Programming Interface for WSN

5.4.1 Introduction

The MSP430 Internet connectivity achieved our objective of implementing a web-enabled wireless sensor network, but the implementation was falling short of many other necessary features. Re-evaluating the design issues and setting higher standards for the web server implementation required an improved hardware and software specification. The following requirements and features describe the reasons for our shift to another microcontroller:

- *Increased size:* The MSP430 is successful in monitoring the network data, but with only 60 KB ROM and 2 KB RAM, collecting and storing data into the microcontroller flash is not feasible. An increased memory is absolutely necessary for the data storing capabilities of the embedded web server. The Rabbit Semiconductor RCM3750 core has 512 KB flash memory and 512 KB static RAM, which enables an increase in the number of web pages stored on the microcontroller. The RCM3750 RabbitCore has a serial flash of 1 MB memory, which makes it a great choice for data collection and storage applications.
- *Increased security:* The web-based user management interface can control the wireless sensor network operations. Because the application allows update of wireless sensor network operations via remote access, it is important to add some access controls and security features to the embedded web server to restrict access to the critical access controls of the web-enabled wireless sensor network. The resource scarcity on the MSP430 required eliminating the security features on the web server, and unlike the MSP430, the RCM3750 RabbitCore is capable of implementing the security features into the web server.
- *SSI and CGI support:* Requesting network parameters from the wireless sensor network required developing CGI and SSI functions to initiate the request and include the dynamic data into the response. The inbuilt support for CGI and SSI functions on the Rabbit3750 makes it easy to develop a web page "on the fly."

- *Support for virtual file system:* The RCM3750 RabbitCore supports a virtual file system, allowing the web pages with images and improvised organization of file system and storage.
- *Support for web compiler:* The dynamic C add-on module, RabbitWeb, is an extension to the 'C' language syntax and it is used to simplify the process of presenting the web variables to the server. With the help of RabbitWeb, the web pages can be made highly dynamic in nature and setting extensive access controls to the device web pages is possible.
- *Dynamic C:* The software developing environment, Dynamic C comes with a royalty free TCP/IP stack library which implements all the basic functions required for a HTTP server, making the application development easy by avoiding the implementation of the TCP/IP stack from scratch.

Exploiting the above advantages over the MSP430, this thesis developed an application programming interface to implement a secure and reliable HTTP server. A set of firmware is developed to monitor and control the wireless sensor network, and is also able to collect the data from the network. The firmware includes the protocol developed to implement the communication between the web server and the network by connecting the web server to one of the wireless sensor nodes of the network. The data collection and retrieving techniques are developed; further, a web-based user management interface is designed and all the web server controls are connected to this interface. The user interface needed to layout forms which allow the user to fill in the information as a request and submit it to the server. The server application performs the requested operation and sends the result to the client with the necessary dynamic content embedded into the web page.

The level of complexity of programming increases greatly with the increase in demand for the dynamic nature of the user management interface. Several other features which should be considered in developing a reliable and robust web server are user authentication and access control, amount of configurable data uploaded to the server,

number of web pages, number of controls on the web page, etc. Each feature needs an application program to handle the requirement.

5.4.2 Application Block

This section discusses all the functions and capabilities of the Rabbit web server. The organization of the rest of the chapter is based on the web architecture described in Section 5.3.2. The application block manages all the capabilities and features of the web server to remotely control and monitor the web-enabled wireless sensor network. Each application block is described in detail, emphasizing the salient features of the Rabbit web server. This part of the section describes the "compile time initialization." The compile time initialization is sub-divided into MIME table, static resource table and program flash.

- *MIME mapping table:* The MIME mapping table indicates the browser how the content is supposed to be presented to the user. The MIME mapping table in web-enabling the wireless sensor network include: web pages of type basic HTML (.html), CGI functions (.CGI), images (.gif), and ZHTML scripting file (.zhtml). The ZHTML scripting files are the basic HTML web pages with the scripting language embedded into the HTML code. When a ZHTML file is requested, the web compiler, RabbitWeb, runs the ZHTML scripted code and embeds the dynamic data into the web page. The web page is then sent to the client as a normal HTML file. The browser is ignorant of the ZHTML to HTML conversion and decodes the web page as a normal HTML page.
- *Static resource table:* The static resource table defines the resources in Dynamic C. The resources here are the external files like web pages, images, etc., that are included in the server code. These resources are included into the server code by calling the external files using a directive #ximport and #zimport. The difference between these two forms of importing the external files are discussed in future sections. The static resource table defines only the resources which do not have any dynamic data to include, as in the case of an image file. The images displayed

- *Dynamic resource table:* The dynamic resource table defines the resources that include the dynamic data. The resources here, are the web pages of the wireless sensor network which hold the dynamic data. All these external files are included into the server program by calling them using #ximport directive. Following are the web pages defined in the dynamic resource table:
    - *Index Page:* The index page holds the network data monitoring controls.
    - *Result Page:* This page holds the result to the monitoring data request made from the index page.
    - *Admin Page:* The administrator web page holds the critical web server controls. These controls are used for setting the mode of operation of the web server.
    - *WSNControls:* This page holds the sensor network configuration controls.
    - *ServerCtrls:* As the name indicates, this page holds the critical web server controls.
    - *SetTime Page:* The time web page holds the controls for setting the date and time on the server.
    - *PrintLog Page:* The print log page is used to display the requested flash session contents.
    - *WSNContorls Page:* The wireless sensor network control page is the user interface to the sensor network control attributes.
    - *EmailAlerts Page:* The email alerts web page holds the conditions for generating the email alerts.

The most common way of adding access to wireless sensor network controls is to define a set of users, add method of authentication, and define the permissions to each resource. The user table and rule table are the relevant components in the web architecture that takes care of the server security features.

- *User Table:* The user table contains the list of user IDs and the authentication information. Each user ID contains a group mask indicating to which user group the user belongs. Sixteen user groups can be defined, and each user can belong to

one or more user groups. Two additional masks are defined in each user table; the first mask indicates the write access capability to each resource and the second mask indicates the server to recognize the user.

- *Rule Table:* The rule table is a list of information generally called permissions, associated with each resource. Each resource has the following information:
    - The realm used by the servers including HTTP
    - The group mask of the user groups that are allowed read-only access
    - The group mask of the user groups that are allowed write access
    - The server groups that are allowed access to the resource
    - The authentication method used to access the resource
    - The MIME of the resource.

  The rule table can define permission to all the resources in the static and dynamic tables. If the files in the resource table are numerous, a simple one-to-one table entry would waste a lot of storage; hence, a prefix matching algorithm can be used to define the permission to all the resources in a single directory. This needs only one rule table entry, provided all the resources have the same permissions.

The "main loop" is a mandatory endless routine which calls the functions of the TCP/IP library. The main loop drives the background processing necessary to handle all the incoming packets. The main loop ensures that the TCP/IP stack has had a chance to process all the incoming packets. The incoming packets are not restricted to only TCP, but also include other protocols such as ARP and ICMP.

The rest of this section describes the application specific I/O and CGI functions. The objectives of web-enabling the wireless sensor network include remotely monitoring and controlling the sensor network and collecting the network monitored data. This requires a web-based user management interface, defining all the possible controls and data collecting options of the sensor network. The graphical user management interface provides a highly interactive interface to access the sensor network.

5.4.3 Communication Protocol between the Server and the Sensor Node

The major requirement for web-enabling a wireless sensor network is to establish a communication between the web server and the wireless sensor network. A communication protocol is developed to establish that communication. There are two ways of connecting the sensor network to the Internet. One is to connect the web server directly to the hub, a central node which monitors the network operations. The advantage of this method is shorter delay in responding to the requests from the server; but connecting the server directly to the hub limits the mobility of the server making it an integral part of the hub. The alternative way is to connect the server to one of the sensor nodes of the network and the communication between the server and the hub is carried by this special node. The special sensor node, henceforth called an embedded web server node, is a regular node with an additional functionality that allows it to communicate with the web server. The latter approach has an advantage: the server can be mobile and can be used in multiple clusters of sensor nodes. The disadvantage of this approach is it increases the delay in responding to the requests from the server (the message has to travel from the embedded server node to the hub and back, thereby increasing the time delay between the request and the response). The generalized picture of the research setup is presented in Figure 1-2. When a client requests the network monitored data over the Internet, the web server issues a request to the embedded web server node which forwards the request to the hub. The hub responds with the data to the embedded web server node which in-turn forwards the data to the server. The server then presents the data to the client using a graphical user interface. Figure 5-7 shows the request and response data flow between the client and the server.

Figure 5-8 shows the simplified frame format of the wireless sensor network. The embedded web server node frame contains the regular sensor node data slots and embedded server slots follow at the end. The detailed communication protocol between the sensor nodes and the hub is outside the scope of this thesis, but the communication protocol between the web server and the embedded web server node is described here.

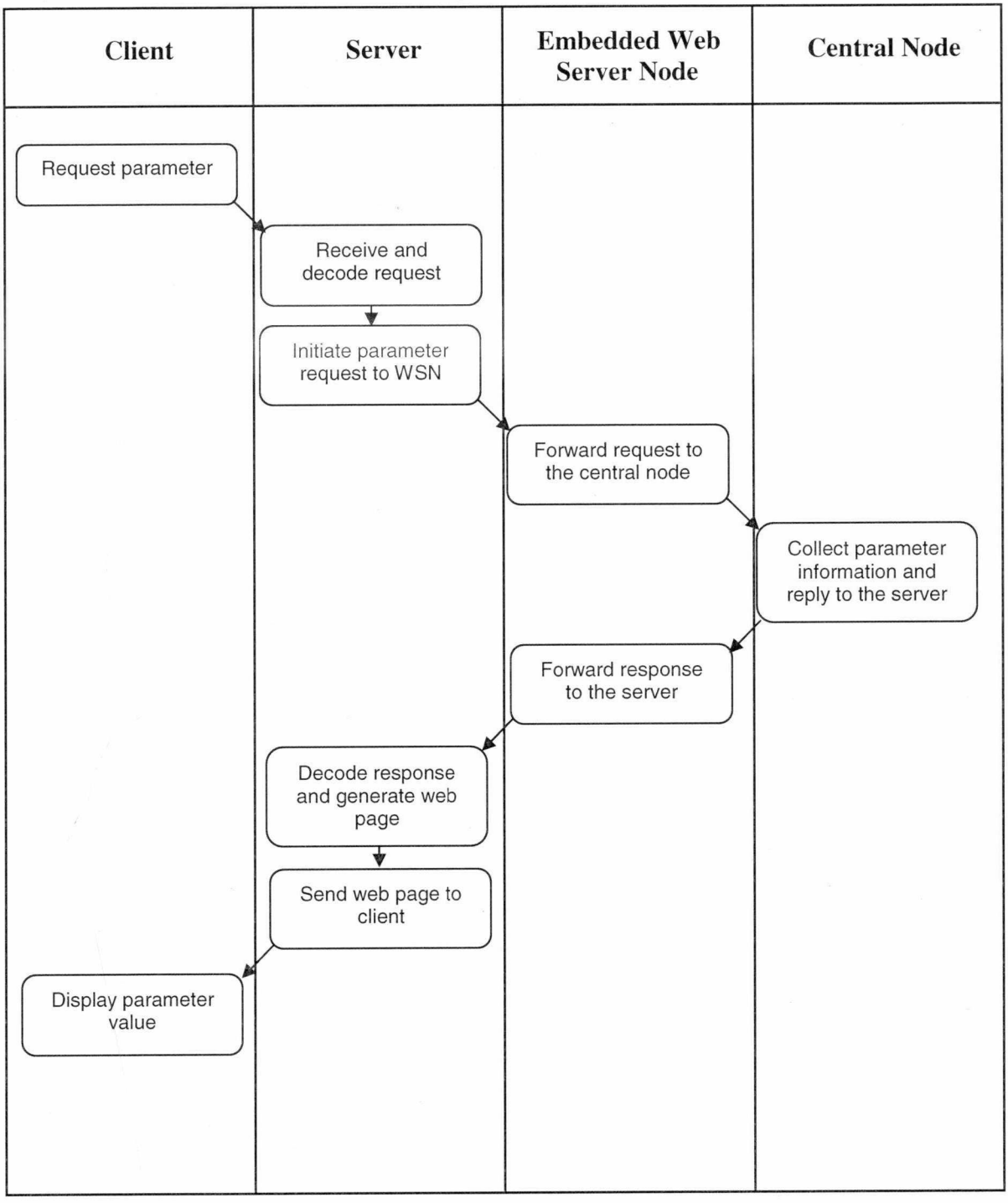


Figure 5-7: Request and response data flow between the client and the server

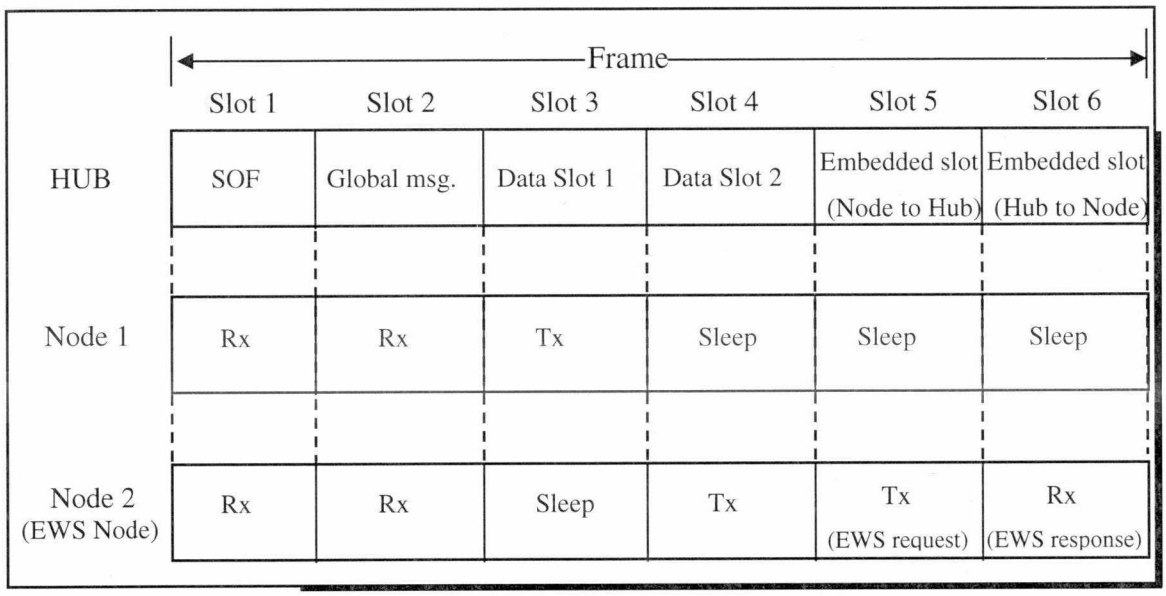


Figure 5-8: Simplified frame format of the wireless sensor network

The communication between the server and the wireless sensor network is defined by a fixed message format. Table 5-1 shows the fields in server and sensor node communication message. The possible examples of communication include request and response for the list of active sensor nodes in the network, and request and response for the collected data.

Table 5-1: Fields in embedded web server and node communication message

Field	**Length in Bytes**	**Purpose**
Sensor ID	1	If the message is a single sensor ID response, this field identifies the sensor ID; If the message is a active sensor list or a parameter request, this field is null
Hub and Message ID	1	The most significant nibble identifies the Hub ID and the least significant nibble identifies the message ID
Payload	12	Parameter data

In the current implementation, the web server can monitor up to four parameters, which can include temperature, battery voltage, vibration, magnetic field strength, humidity, etc., corresponding to any sensor node in the network. The number of parameters and their size can be extended to accommodate the wireless sensor network. Each message is identified by a unique hub and message ID. Table 5-2 gives the list of message IDs used by the server to communicate with the sensor network. The message description with an asterisk (*) indicates a WSN control message.

Table 5-2: Message IDs used by the server to communicate with the sensor network

Message Description	**Message ID (Hex)**
Request/response of list of active sensor nodes	0x0C
Request/response of sensor payload	0x04
Request/response of sensor node radio bit rate*	0x01
Request/response of sensor node radio address bytes*	0x02
Request/response of sensor node radio output power*	0x03
Request/response of sensor node radio frequency channel*	0x05
Request/response to shut down a sensor node*	0x06
Request/response to monitor selected parameters*	0x0A
Request/response of sensor network time between frames*	0x09

The following section presents the message formats of all possible messages exchanged between the server and the embedded web server node.

- *Message format to update the list of active sensor nodes in the network*

A request for the active sensor list is sent to the central node regularly to update the list of sensor nodes currently active in the network. The response from the central node is encoded in order to include the entire list of active sensor nodes in a single message. Each bit in the message data byte represents the corresponding sensor node active/not-active in the network. A bit set (indicated by 1), indicates that its corresponding server node is present in the network, and conversely a bit not set

(represented by a zero) indicates that the sensor node is not present/ not active in the sensor network. A request for the list of active sensor nodes in the network is identified by the message ID and rest of the message fields can be null. Figure 5-9 presents the message format of active sensor list request. Hub ID is 0x10 and the message ID is 0x0C. Figure 5-10 presents an example of active sensor list response message format.

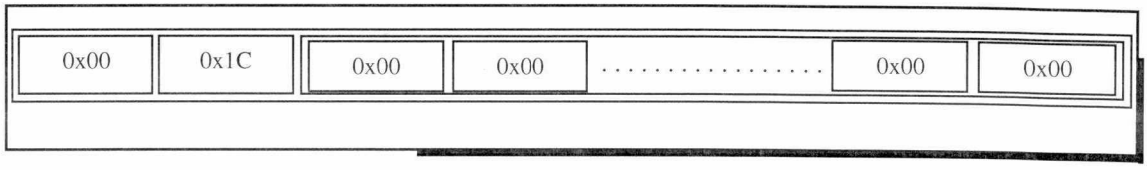


Figure 5-9: Active sensor list request message

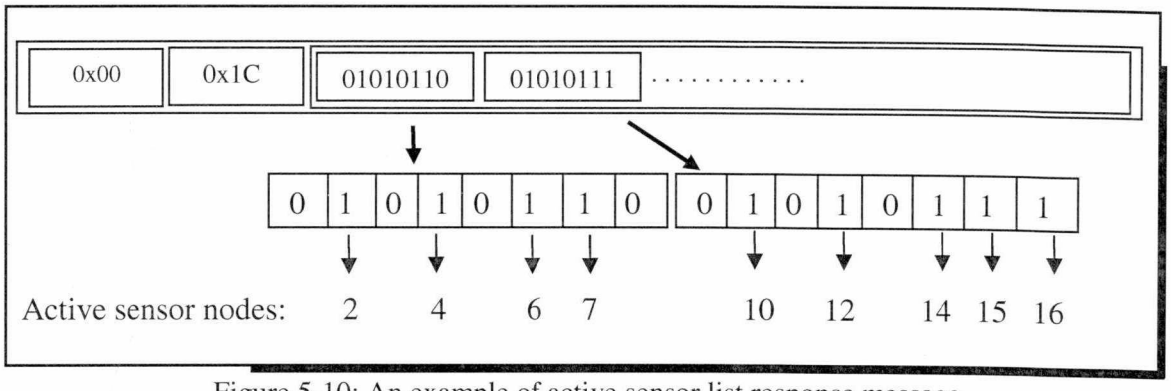


Figure 5-10: An example of active sensor list response message

Once the active sensor list is available on the web server, the user can monitor the network data. The server is capable of requesting parameter data of either a single sensor node or all the sensor nodes active in the network. A single sensor node or multiple sensor nodes parameter data request is sent using a single message with the sensor ID(s) encoded in the payload field. The response to the multiple sensor nodes parameter data request is received in individual messages, each with a sensor ID and corresponding data.

- *Message format of request/response of parameter data*

A request for data of either a single sensor node or all the active sensor nodes in the network is sent in a single message. The request for data of either a single sensor node or all active sensor nodes is coded exactly as seen in the case of active sensor node response. Figure 5-11 and Figure 5-12 show the request and response of single sensor node data.

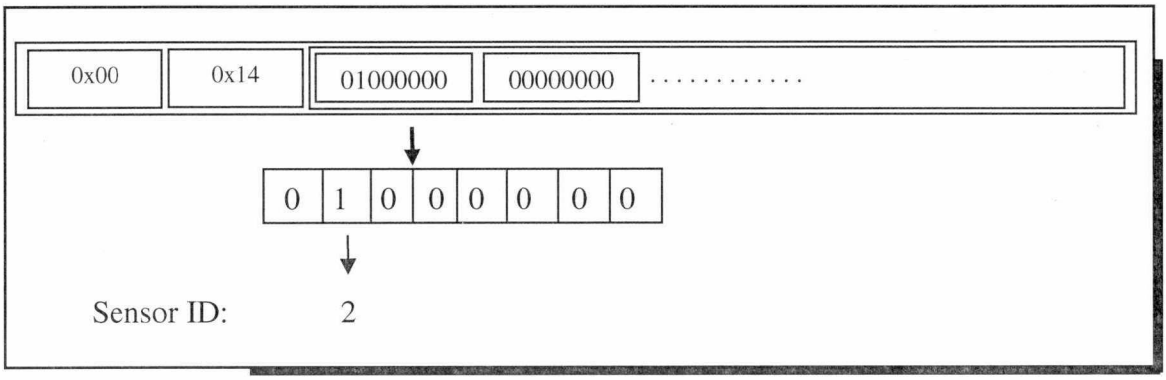


Figure 5-11: Single sensor node data request message

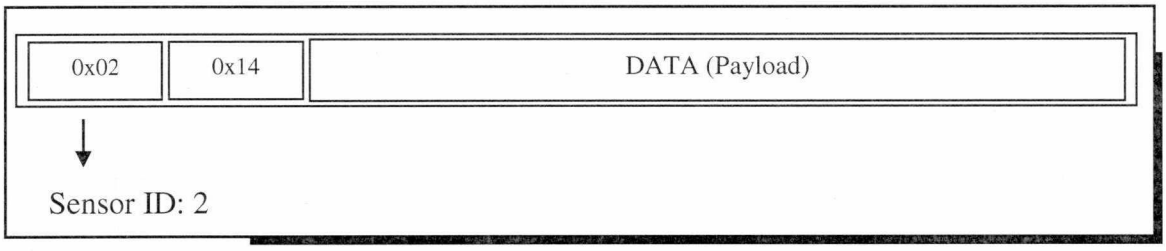


Figure 5-12: Single sensor node data response

Figure 5-13 and Figure 5-14 shows the request and response of all active sensor nodes data. The multiple sensor nodes parameter data request is sent in a single message, but the response is sent in individual messages.

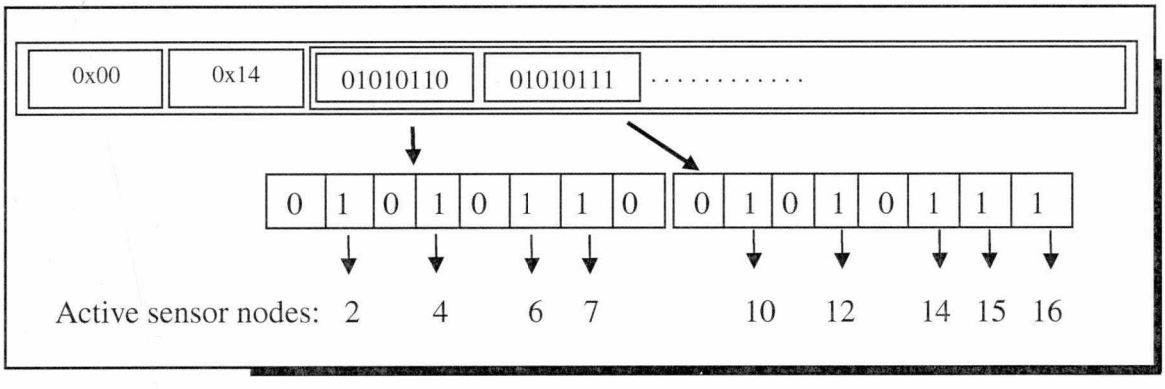


Figure 5-13: Multiple sensor nodes data request

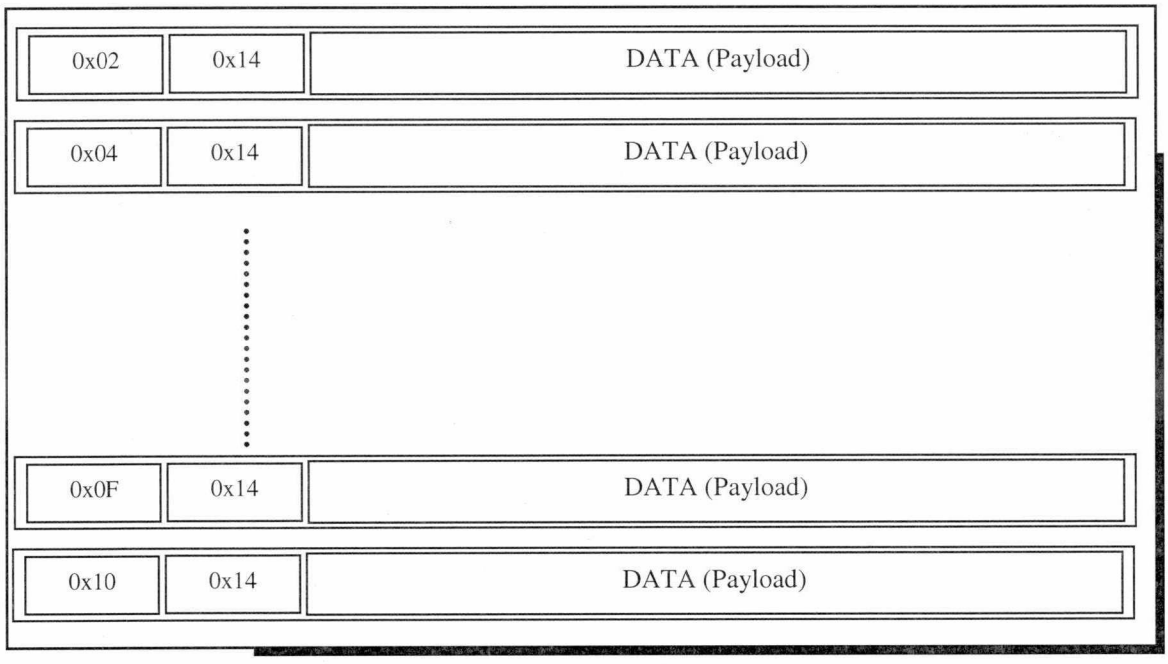


Figure 5-14: Multiple sensor nodes data response

- *Message format to change the sensor network frame period*

The administrator can configure the time between frames of the sensor network. The server expects the frame period (FP) from the administrator in 100's of milliseconds. The value is forwarded to the sensor network by embedding it into the message with network frame period message ID, Figure 5-15 shows the message format.

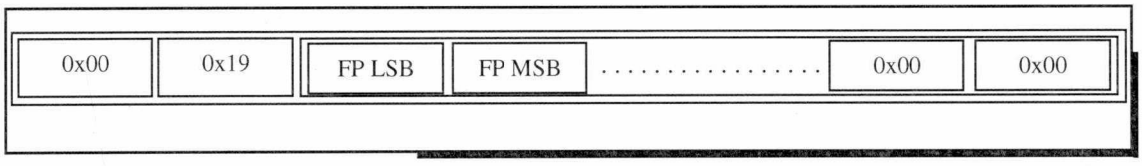


Figure 5-15: Message format to update the time between frames of the sensor network

- *Message format to update the list of network monitored parameters*

The server is capable of configuring the network to monitor four parameters. The administrator can configure the network to start/stop monitoring any parameter. When the administrator modifies the parameter list in the WSN controls web page, the server sends the request with the corresponding bit set or unset (1 indicates to start monitoring the corresponding parameter and 0 indicates otherwise) that are supposed to be monitored by

the network. Figure 5-16 shows the message format to modify the list of parameters monitored by the sensor network.

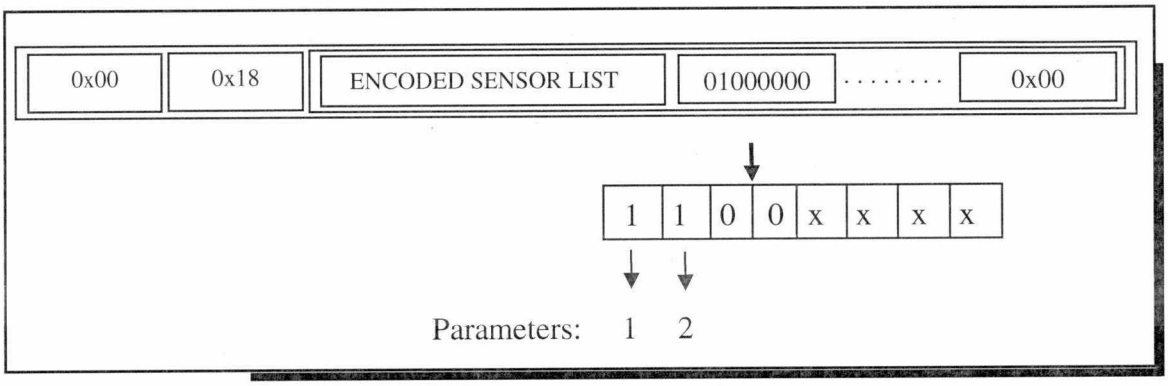


Figure 5-16: Message format to update the list of network monitored data

- *Message format to shutdown any active sensor node*

The administrator can request to shut down any active sensor node in the network. When a particular sensor node is requested to turn off, the server sends the shutdown request with the encoded bit stream of sensor ID embedded into the payload of the message format. After the shutdown request is sent to the network, the server waits for pre-defined duration of time and requests the list of active sensor nodes to update the active sensor list on the server. Figure 5-17 shows the message to shutdown an active sensor node.

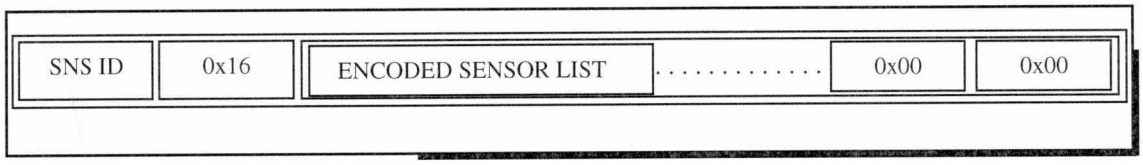


Figure 5-17: Message format to shutdown a sensor node

- *Message format to change the sensor node radio output power*

The possible output power configurations allowed in the radio used on the current sensor node are -20 dBm, -10 dBm, -5 dBm, and 0 dBm. The central hub receives the value and configures the output power of the radio. Figure 5-18 shows the message to change the node radio output power.

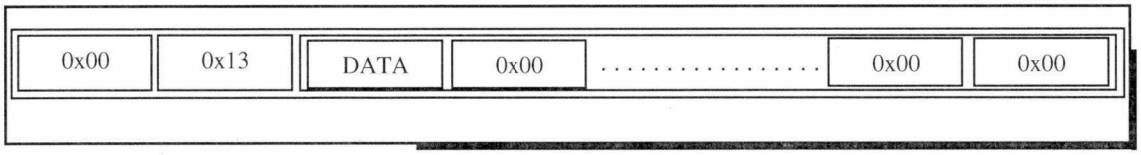


Figure 5-18: Message format to change the radio output power

- *Message format to change the sensor node radio bit rate*

The possible bit rates (BR) configurable on the sensor network radio are 250 kbps and 1000 kbps (1 Mbps). The server sends the bit rate value in kbps. Figure 5-19 shows the message to change the sensor node radio bit rate.

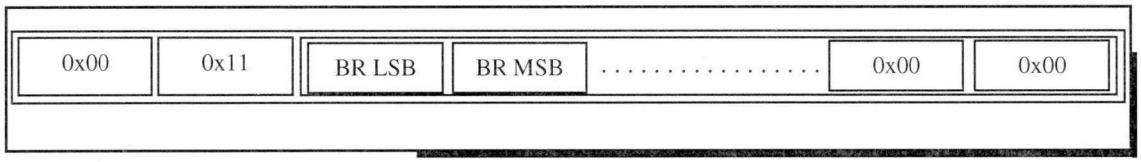


Figure 5-19: Message format to update the sensor node radio bit rate

- *Message format to change the sensor node radio frequency channel*

The sensor node radio frequency channels range from 2400 to 2482 MHz with 1 MHz apart. When the administrator changes the sensor node radio frequency channel, the server sends a single byte with values ranging from 00 to 82 (last two digits of the channel number). Figure 5-20 shows the message format to change the node frequency channel.

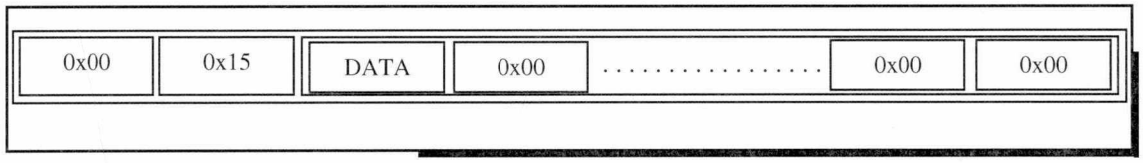


Figure 5-20: Message format to change the sensor node radio frequency channel

- *Message format to change the sensor node radio address bytes*

The possible number of address bytes configurable on the sensor node radio range from 1 to 5. When the administrator changes the sensor node radio address bytes, the server forwards the request with the address size byte embedded into the payload of the message. Figure 5-21 shows the message format to change the sensor node radio address bytes.

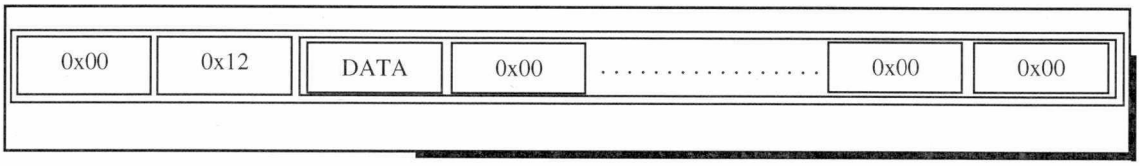


Figure 5-21: Message format to update the sensor node address bytes

5.4.4 Server – User Groups

The web server access controls are greatly extended by adding three different set of user groups. The "open to all" user is a highly restricted web client and does not have access to the web server controls and the sensor network controls, but the user can view the HTML pages served by the server and can monitor the data collected by the sensor network. A registered user can also change the mode of operation of the server (the server's modes of operations are discussed in the following sections) and can also request previously logged data. The third set of user groups, "administrators" has an unrestricted access to both the server and sensor network controls.

The rule table defines the permission to the web server resources. The resources defined in the static and dynamic resource tables which hold critical information of the server operations include:

- *Server configuration web page:* This page is used to set the web server mode of operation.
- *Email alert page:* This web page is used to set the conditions for sending email alerts.
- *Date page:* This web page is used to set the date and time of the server.
- *Log request/response page:* This web page holds the log data display controls.

The sensor network resources include the network configuration web page, which holds the controls of wireless sensor network.

The rule is added to the directory "/Monit/" with the realm name "Server Controls." Read access is allowed for registered users and write access is not necessary as the web pages are not supposed to be modified. Here, the write access is defined for the web pages itself, not for the controls on the web pages. The resource under this rule is the

server configuration web page. The server configurations include the controls for setting the mode of operation of the server and requesting the logged data. The resource can be accessed by any registered user and "basic authentication" method is used (another possible authentication method is "digest," here the password is encrypted unlike in basic authentication where the user password is not encrypted). This user group is helpful for creating a set of users with an access to the sensor network, for example in the case of student practical training for a wireless sensor network course. The server is programmed such that only a single registered user can login to the server at a time. Once the user logs off the server, another registered user can login to access the server controls. The server is also programmed to automatically log off the user after a specified duration of inactivity to avoid permanently holding the access to the server by a single user.

The rule is added to the directory "/Admin/" with the realm name "Administrative Controls," read access is allowed to all the server resources - the server configuration web page and the sensor network configuration page. All the critical controls of the web server are defined in a single directory named "/Admin/" thereby setting common permissions to all the web pages.

5.4.5 Administrative Controls

The administrator has unrestricted access to all the server controls. Besides configuring the operating mode of the server, an administrator has an access to several other web server and sensor network controls. The following subsections present some of the controls of the web server, and Appendix B includes the snapshots of the web server administrative controls.

5.4.5.1 Web Server Controls

The web server controls include resetting the operating mode of the server, clearing the log, configuring the email alerts, and setting the date of the server.

- *Operating Modes:* The server can be operated in three different modes of operation, which include Monitoring mode, Log mode, and Snapshot mode. The

administrator can set the server into any of the above operating modes (Figure B-4). The following section explains these operating modes in detail.

- *Clearing the Log:* The administrator can clear the logged data, thereby allowing the server to start logging the data afresh.
- *Email Alerts:* The server is capable of sending an email to an administrator or an authorized email user from the web server. The email alert configuration page (Figure B-12) sets the conditions for sending the mail. The configurations include sending an email in the following cases:
    - If the server does not respond to configured number of requests from the web server.
    - If the server's serial flash is full.
    - If the sensor parameter values reach the set threshold.
    - In the wireless Internet mode, the server sends an email to the administrator with the DHCP assigned dynamic IP address to access the server.

  The server includes the appropriate subject and body of the email describing the reason for the email alert (Figure B-13). In a case when one or a list of the sensor nodes reaches the set threshold, the server includes a dynamically generated email body identifying the sensor node(s) which reached the set threshold. This feature keeps the administrator informed about the network status and helps to keep track of the network monitored data.
- *Setting the Server Date:* The administrator can set the date on the server. The date on the server is used to record the time when a particular session is logged in.
- *Log local temperature:* The administrator can log the local temperature of the EWS. The server requests the temperature from the embedded sensor node and logs the data. Once the data is collected for the configured number of samples, the server sends an email to the administrator with the session number of the log.

5.4.5.2 WSN Controls

The administrator can also configure critical controls of the sensor network including the sensor nodes radio configurations and network data configurations. Appendix B includes a snapshot of the WSN configuration web page.

- *Radio Configurations:* In our present case study, the embedded web server monitors an energy efficient wireless sensor network. Radio configurations are one of many design configurations considered to implement an energy efficient application specific WSN. The web server is capable of configuring these radio configurations which include radio output power, bit rate, communicating channel, etc.
- *Network Configurations:* The server is capable of configuring the network data configurations which include monitoring selected parameters, time between frames, and shutting down a particular sensor node.

5.4.6 Server – Operating Modes

The web server has different operating modes to fulfill the objectives of web-based data collections and monitoring of network parameters. The server configuration page holds all the controls which relate to the web server's mode of operation. The web server can be operated in any of following three modes: snapshot mode, monitoring mode, and log mode. Appendix B includes the snapshots of the web server web pages working in all the three modes, and the following sections explain these operating modes.

5.4.6.1 Snapshot Mode

In this mode, the server passively listens to the requests for network parameters from the clients. When the client requests any parameter, the server issues the request to the embedded web server node. The embedded web server node forwards the request to the central node. The central node retrieves the last collected network parameter data and responds to the embedded web server node which in turn forwards it to the server. The server processes the received network data and presents the graphical representation of

the parameter value to the client. Figure 5-7 shows the data flow between the client and the server in snapshot mode. Snapshot mode is used to get the current parameter values monitored by the network. It gives the glimpse of the monitored network data.

5.4.6.2 Monitoring Mode

In this mode, the server takes the administrator's choice of parameters to continuously monitor the network and actively collect the data from the network. When the server is operating in monitoring mode, the result shows continuous live parameter values using a graphical representation of the data. The time between the requests depends on the frame period of the wireless sensor network. Figure 5-22 shows the simplified flowchart of the server operating in monitoring mode.

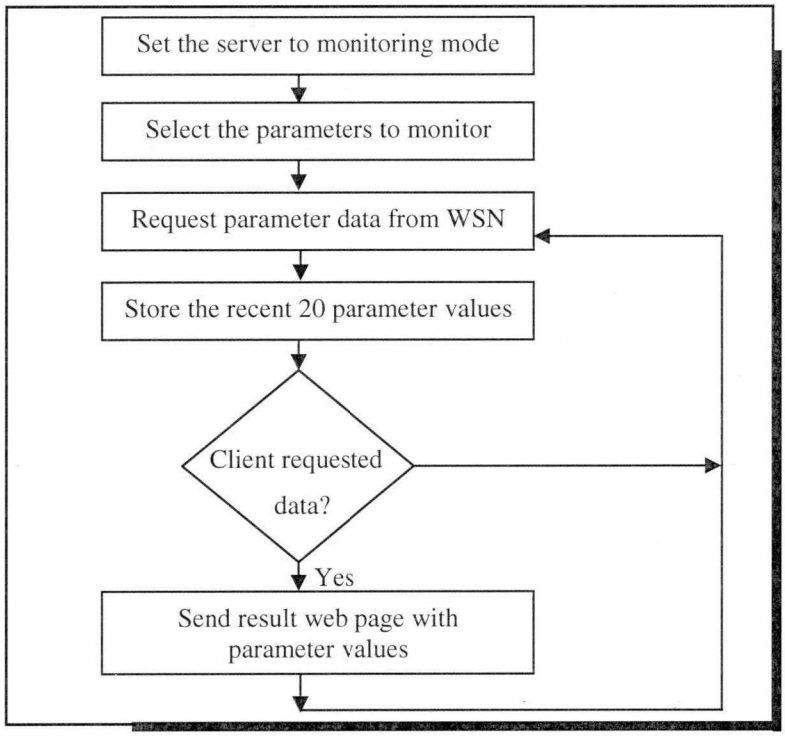


Figure 5-22: Generalized flowchart of the web server in monitoring mode

5.4.6.3 Log Mode

In this mode the server, using the administrator's choice of parameters, collects and logs the data from the sensor network. The server collects the data for the specified

number of samples and stores it on the serial flash memory of the web server. The data can be remotely read from the web server's flash and the parameter values can be decoded from the data using an application developed in MATLAB®. The flash contents can be read either from the server's web pages through the Internet browser or directly from the serial port of the web server. This section discusses the log mode of operation in detail.

The RCM3750 has an Atmel Flash AT45DB081B, a serial interface 1 MB flash memory. Figure 5-23 shows the block diagram of the Atmel's data flash memory. The device consists of a flash memory array, two data buffers, and an I/O interface. The flash memory array compromises of 4096 equal length pages, 264 bytes each. Each page has 5 bytes reserved for control information: page pointer and data length. This control information, which is vital to the systems operation, is used to determine how to utilize the pages for reading and writing data into flash.

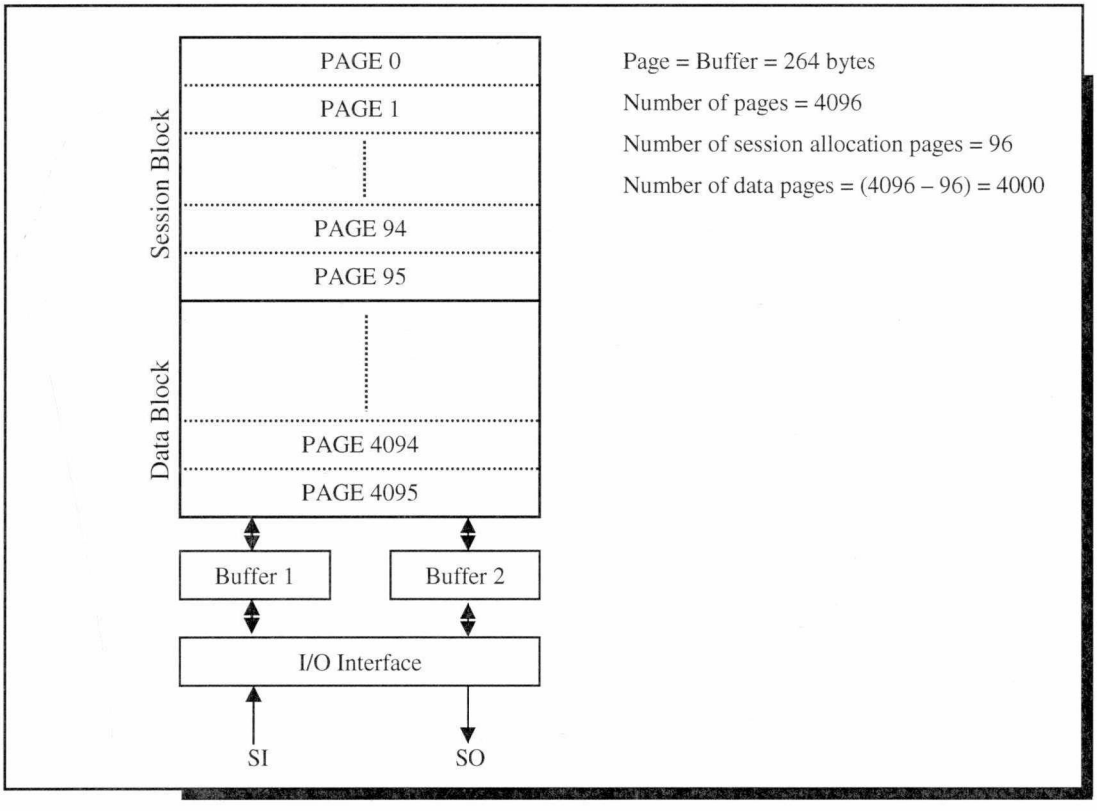


Figure 5-23: Serial flash memory architecture block diagram

The flash memory array is divided into two blocks: session allocation block and data block. Session allocation block contains the session allocation table (SAT) which stores the page number and the displacement within the page where the session is stored. Data block contains the parameter data and its associated configuration fields. Storing data into flash requires storing the actual parameter data and its associated configuration. The configuration of a particular session includes the time when it is captured, the active sensor nodes present in the network, list of parameters which are captured, message size of individual parameters, and time between each data capture. Table 5-3 shows the fields in a session stored into serial flash. The length of coded sensor list and data fields is variable as the number of bytes for the coded sensor list is defined by the number of sensor bytes, second field discussed in this table. The data field length depends on message sizes and number of samples collected.

Table 5-3: Fields in a session stored into serial flash

Field	**Length in Bytes**	**Purpose**
Time stamp	7	Time when the data capture is initiated
Number of sensor bytes and parameter list	1	The most significant nibble defines the number of coded sensor list bytes and the least significant nibble defines the list of parameters for which the data is captured
Parameter message size	2	The message size of all the individual parameters
Frame period	1	Time between two consecutive data captures
Coded sensor list	Variable	Coded list of sensor nodes active/present in the network
Data	Variable	The actual parameter data captured

The serial flash incorporates two on-chip, bi-directional buffers to enable the data flow to and from the device. The size of buffers is equal to the flash memory array page size, and they function independently. When the server is set to operate in Log mode, the

server writes the session configuration details into the log followed by the actual parameter data. Once all the desired data is logged in successfully, the session details are appended to the session allocation table. Figure 5-24 shows the flowchart of data log operation.

Once the data is logged, the client can read the data directly from the server through standard serial communication interface or remotely through the web page. A MATLAB® application is developed to decode the session details and parameter values from the logged data. The application is an interactive graphical user interface and it replicates the web page browser controls to present the decoded data.

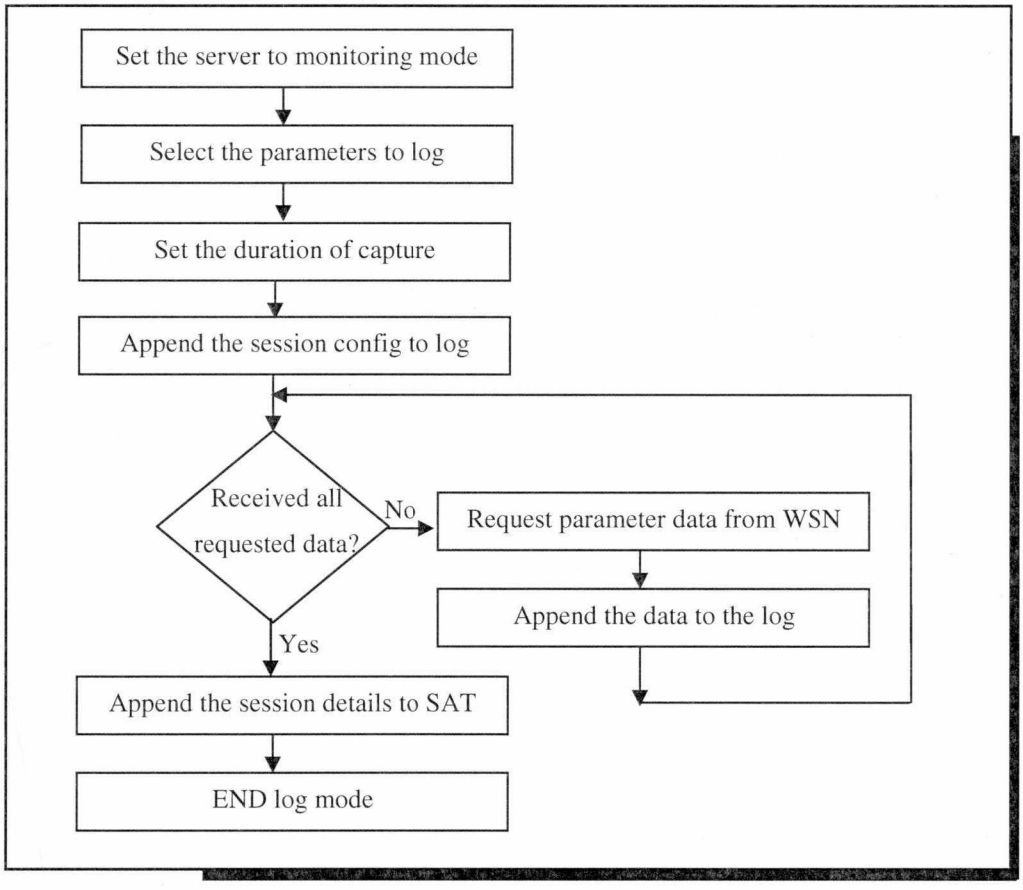


Figure 5-24: Flowchart of logging data in log mode

5.4.7 Server-side Programming

Server-side programming is used to create web pages for the WSN that interface to an Internet browser. Server Side Include (SSI) and Common Gateway Interface (CGI) are server side methods of including the parameter data into the HTML document. SSI and CGI are set of functions built into the web server that gives it an ability to insert data into HTML documents using special directives. A plain HTML document that a general web server retrieves is static, which means that the page exists in a constant state. On the other hand, Rabbit web server serves dynamic information using the server-side programming. This allows the server to create dynamic documents with the parameter values embedded into the normal HTML pages.

SSI works by configuring the web server to parse through a normal HTML document looking for SSI variables. When a SSI variable is found, the server carries out required instructions and replaces the variable with the generated output. RabbitWeb, an add-on module to the Dynamic C programming language is a scripting language like SSI which is used to embed dynamic content into web pages. We used both SSI and RabbitWeb to dynamically generate the web pages. The Rabbit web server hosts both static and dynamic web pages. The following web pages in our design use SSI, RabbitWeb, or both to dynamically insert the data into the document:

- *Index Page:* The index page holds the network data monitoring controls. The index page is dynamically generated depending on the web server's operating mode (different operating modes have different index pages). In snapshot and monitoring mode, the active sensor list is updated according to the network.
- *Result Page:* The result web page holds the result to the data request made from the index web page. Similarly to the index page, the result page is dynamically generated depending on the server's mode of operation.
- *Monit Page:* The administrator web page holds the critical web server operating controls. The server embeds the serial number of the latest session that was successfully logged into the web page.

- *SetTime Page:* When an administrator requests the time page, the server inserts the current date and time on the server into the web page. The administrator can also modify the time and date on the server.
- *PrintLog Page:* The server inserts the log contents of the requested session into this page.
- *EmailAlerts Page:* The email alerts web page holds the conditions for generating the email alerts. The server inserts the preset email alert conditions into the web page and an administrator can modify these conditions to generate an alert.

The web page file with the extension ".zhtml" indicates that it would be parsed by the server before being sent to the client. If a zhtml instruction is printed on the browser then the web page is probably not being parsed for the "includes," or an error occurred during the scan. SSI basically dumps the contents of a web variable straight into an HTML document. Alternatively, CGI allows the web server to actually run a program and return the output to the client. A CGI function is executed in real-time to output dynamic information. The main difference between SSI and CGI is that an SSI function is executed when a web page is sent to the client and a CGI function is executed when a web page is requested. Following are some of the CGI functions written to web-enable a wireless sensor network:

- *ResetOpMode* is a CGI function that is executed when the administrator wants to reset the web server's operating mode.
- *RefreshSL* is a CGI function that is called to send a request to the sensor network for the list of active sensors in the network.
- *ResetLog* is called when the administrator wants to clear the logged data and start the logging process afresh.
- *CopyGraphVals* is called when the client requests network data while the server is operating in monitoring mode.
- *UART_Request* is called when the client requests network data while the server is operating in snapshot mode.

- *PrintLog* function reads the contents of flash and redirects the client to the print log web page.

The above functions are some of the examples of many CGI functions that are written to web-enable the wireless sensor network. Server-side programming describes the scripts and functions that are executed by the server when a client requests a dynamic web page. Server-side scripts (CGI functions) and server pages (SSI) are the server-side programming types used on the Rabbit web server to web-enable the sensor network.

5.4.8 Client-side Programming

This section describes the CGI programming where the processing takes place on the client machine rather than on the web server. As discussed earlier, a plain HTML document is static and exists in a constant state. The browser displays the text and graphics and waits for the client to click a link or input some data. From the web client's view, dynamic capabilities of the web documents are limited mainly to clicking a link or submitting an input to the web server to receive more static web pages. Client-side programming is developed to add dynamic behavior to the browser by executing programs on the client's machine.

Client-side programming describes the functions/scripts that are executed by the browser. It is required for good visual effects and form validation. The scripts are typically mixed with the HTML code of the web document. HTML provides the user-interface and the scripts provide the logic to govern the dynamic behavior of the document. In this project, the client-side programming, JavaScript, is used to create a dynamic, interactive web-based application that completely runs within a browser. The following are some web pages that are embedded with Java Scripts to add dynamic behavior to the browser:

- *Admin page:* Highly interactive and user friendly controls are developed for the administrator web page to set the server's mode of operation. An intelligent client-side programming is developed to guide the user in setting the operating mode. The client-side programming, JavaScript, provides an HTML level interaction with the user and makes sure that the user has filled all the required

fields for setting the web server's operating mode. Instead of forcing the server to do the data validation, requiring data exchanges between the client and the server, the load on the server is reduced by allowing all the calculation work to be handled by the user's computer. JavaScript is embedded into the web pages to handle all the requirements to set the web server's operating mode. To operate the server in the log mode, the server requires the administrator's choice of parameters to log the data and the number of samples to log. For the monitoring mode, the server requires only the administrator's choice of parameters to monitor. For the snapshot mode, the server does not require any supporting attributes. JavaScript makes sure that the user entered all the required fields for setting the server's mode of operation.

- *Index page:* JavaScript in snapshot mode index page disables the single sensor node request when "all sensors" radio button is selected. The single sensor node request is enabled when single sensor ID is selected.
- *Result page:* JavaScript in monitoring mode result page, display a time counter that counts down from the time between frames. When the time expires, the Java Scripts automatically sends another request for the latest twenty samples that are captured.
- *Email configuration page*: JavaScript validates the user input. One of the examples of the user input includes the email address entered to send the email alerts.

Using JavaScript to add dynamic features to web pages, such as reacting to user events, comes with some disadvantages. In a worst case scenario, a browser with disabled JavaScript cannot display the page correctly. Subtle differences in browser implementations can cause some errors, crippling the user-interacting capabilities of the web page. This possible problem is restricted to that particular client machine and does not affect the web server operations or web page display on other client machines.

5.4.9 Wireless Internet Connectivity

RCM3750 RabbitCore has an add-on kit which allows adding an 802.11b wireless Internet connectivity to the web server. 802.11b is an extension to 802.11 that describes media access and link layer control for 2.4 GHz implementation at a maximum bit rate of 11 Mbps. The wireless connectivity eliminates Ethernet cables, allowing greater flexibility and mobility to the web server. 802.11b allows the web server to operate in one of the two modes: a managed-access mode or an unmanaged mode.

A managed-access mode, also called an infrastructure mode, is the basic service setup where one or more access points are connected to the wired network. The access points provide a wireless connection, allowing the host to link to the wired network. The Rabbit web server can be programmed to operate in infrastructure mode, which allows the client to access the server from a remote location. The unmanaged mode, also called an ad-hoc mode, is a peer-to-peer mode which allows creating a small group of wireless devices. The Rabbit web server can also work in the ad hoc mode allowing a small group of people to establish the wireless sensor network without the infrastructure. Figure 5-25 shows the snapshot of web server operating in ad-hoc mode.

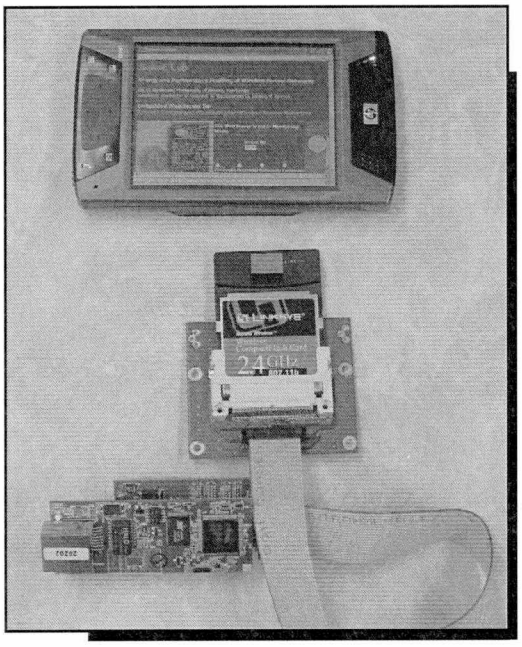


Figure 5-25: Rabbit web server working in the ad hoc mode

If the Internet infrastructure exists in the vicinity of the wireless sensor network, the Rabbit web server can be operated in the infrastructure mode. In this mode, the Rabbit web server either issues a request to the DHCP server for an IP address or uses a previously assigned static one. If the DHCP server is used, the Rabbit web server sends an email with the dynamically assigned IP address to the authorized client through which the web clients can remotely access the sensor network. If the sensor network has no wireless Internet connection available, the Rabbit web server automatically falls back to a private network IP address through which an 802.11b enabled device can remotely access the sensor network. The latter mode of operation is suitable for periodic control or data collection visits to the remote wireless sensor network sites.

6 Conclusion and Future Work

6.1 Conclusion

The widespread acceptance of online communication is increasing the number of appliances and devices connected to the Internet. This acts as a strong drive for the use of embedded web servers in all devices.

Web-enabling a device potentially extends the functionality of the device. A web client or user with a standard Internet browser can see the web-enabled device as a web site. The user can control the device through the Internet and the device responds to the web page buttons, hot links, and other familiar browser controls.

This thesis explores the importance of the embedded web server and presents a case study of web-enabling a wireless sensor network. The wireless sensor network is controlled and monitored, and the data collected by the sensor nodes is accessed remotely through the Internet. In this thesis, the basic Ethernet protocols and different TCP/IP stack implementations are explored and modified to suite the needs of the embedded web server. The technical concepts, including the functional and non-functional requirements of the embedded web server are outlined, and a web-based user management interface is developed to control and monitor the sensor network. The web-based user management interface provides a highly interactive, enhanced, and powerful user interface without any additional hardware at the client site.

The security enabled embedded web server reports the critical events of the sensor network, including the abnormal operation of the network, web server data storage limit, etc., to an authorized user. Reporting the critical information of the wireless sensor network helps the sensor network administrator with the attention required to control the network.

The wireless Internet connection to the embedded web server adds an extra advantage by adding mobility to the web server.

6.2 Future Work

The organization of the collected network data stored on the embedded web server can be improved by implementing a File Allocation Table (FAT) file system. The security and file access capability can be further improved by implementing the SSL. The web server is capable of receiving and handling large amounts of data, typically around tens of kilobytes, and this feature can be used to send the program code to update the sensor nodes firmware. In this thesis the Internet connectivity is implemented using wired and wireless Ethernet connections. Extending the web server connectivity further, for example, using a packet-based protocol for data transfer in cellular phone networks, would be a good add-on feature for increased mobility. The changes to the applications would be minimal, but the new interface between the web server and the Internet would be required.

This thesis is a case study of web-enabling a wireless sensor network; similarly, the web server can be used to web-enable any device and connect the device to the Internet. The web-enabled device then can be remotely configured, controlled and monitored.

7 References

[1] V. Chandrasekar, J. A. Morgan, "Embedding Web Access Mechanism in an Appliance for User Interface Functions Including a Web Server and Web Browser," U.S. Patent 5,956,487, September 21, 1999.

[2] Rabbit Semiconductor, "An Introduction to TCP/IP – For Embedded System Designers," July 2002 http://www.rabbitsemiconductor.com/documentation/docs/manuals/TCPIP/Introduction/tcpintro.pdf

[3] J. Postel, "Internet Protocol (IP)," RFC791, September. 1981. http://www.faqs.org/rfcs/rfc791.html

[4] D. Plummer, "An Ethernet Address Resolution Protocol (ARP)," RFC 826, November 1982. http://www.faqs.org/rfcs/rfc826.html

[5] J. Postel, "Internet Control Message Protocol (ICMP)," RFC792, September 1981. http://www.faqs.org/rfcs/rfc792.html

[6] Texas Instruments, "MSP430 Internet Connectivity Application Report," February 2004.

[7] J. Postel, "User Datagram Protocol (UDP)," RFC793, September 1981. http://www.faqs.org/rfcs/rfc793.html

[8] J. Casad. (2003, September) "*Sams Teach Yourself TCP/IP in 24 Hours*,"

[9] Blunk Microsystems, Date Retrieved: September, 2006, http://www.blunkmicro.com/webreasons.htm

[10] J. Blaisdell, "Designing a Web-based Management System the Right Way," Date Retrieved: September, 2006, http://www.intel.com/technology/magazine/computing/ac09991.pdf

[11] J. Hong-Taek, C. Mi-Joung, and J. W. Hong. (2000). "An Efficient and Lightweight Embedded Web Server for Web-based Network Element Management," *International Journal of Network Management*. 10, pp. 261-275.

[12] A. Imran, W. Hong and K. Vikram. (2004, July). "Internet Based Remote Control using a Microcontroller and Embedded Ethernet Board," *Proceedings of American Control Conference*. 2, pp. 1329-1334.

[13] NetBurner Inc., Date Retrieved: September, 2006, http://www.netburner.com/

[14] Basic Stamp 2 (BS2P40) microcontroller –Parallax Inc., Date Retrieved: September, 2006. http://www.parallax.com/detail.asp?product_id=BS2P24-IC

[15] A. Turan. (2006, August) "Remotely Accessible Hardware-in-the-Loop Robot Simulator."

[16] K. Tim, B. John, M. Jeff, B. Gene, C. Debbie, D. Philippe, G. Gita, F. Marcos, K. Venky, M. Howard, S. John, S. Bill and M. Spasojevic. (2004, October). "People, Places, Things: Web Presence for the Real World," ACM MONET (Mobile Networks & Applications *Journal).* 7(5), pp. 365-376.

[17] R. Janne, M. Peri, M. J. Saaranen, R. Jussi and S. Juha-Pekka. (2001, October). "Providing Network Connectivity for Small Appliances: A Functionally Minimized Embedded Web Server," *IEEE Communication Magazine.* pp. 74-79.

[18] Blunk Microsystems – TargetWeb, Date Retrieved: September, 2006, http://www.blunkmicro.com/web.htm

[19] GoAhead Software – Web Server 2.1, Date Retrieved: September, 2006, http://webserver.goahead.com/webserver/webserver.htm

[20] Mbedthis - AppWeb HTTP Server, Date Retrieved: September, 2006, http://www.mbedthis.com/

[21] AllegroSoft – RomPager, Date Retrieved: October, 2006, http://www.allegrosoft.com/company.html

[22] BVM – IntraScada Web Server, Date Retrieved: October, 2006, http://www.bvm.co.uk/intrascada_ds.html

[23] Quiotix – Quiotix Embedded Web Server, Date Retrieved: October, 2006, http://www.quiotix.com

[24] Moteiv – Tmote Connect, "Tmote Connect – Data Sheet," February 2006, http://www.moteiv.com/products/docs/tmote-connect-datasheet.pdf

[25] WebCoffee project, Date Retrieved: October, 2006, http://www.udel.edu/PR/UpDate/99/19/web.html

[26] H. Kamal and P. Bob. (2004, November). *Embedded Systems and Design Using the Rabbit 3000 Microprocessor: Interfacing, Networking and Application Development.*

[27] Olimex Limited., Date Retrieved: September, 2006, http://www.olimex.com/

[28] Texas Instruments, "MSP430x1xx, family user's guide," rev. 5, 2006, http://focus.ti.com/lit/ug/slau049f/slau049f.pdf

[29] Cirrus Logic, "CS8900A Crystal LAN™ Ethernet Controller Product Data Sheet" September 2004, http://www.cirrus.com/en/pubs/proDatasheet/CS8900A_F3.pdf

[30] B. Hossein. (2004). *The Internet Encyclopedia.*

[31] Rabbit Semiconductor, "RCM3750 RabbitCore Top and Bottom View," August 2005. http://www.rabbitsemiconductor.com/products/rcm3750/largeView.shtml

[32] Rabbit Semiconductor, "RCM3750 RabbitCore Data Sheet," August 2005. http://www.rabbitsemiconductor.com/products/rcm3750/rcm3750.pdf

[33] Rabbit Semiconductor, "Rabbit 3000 Microprocessor User's Manual," October 2005. http://www.rabbitsemiconductor.com/documentation/docs/manuals/Rabbit3000/UsersManual/R3000UM.pdf

[34] ASIX, "AX88796L Ethernet Controller Data Sheet," rev. 2.3, http://www.asix.com.tw/FrootAttach/datasheet/Ax88796.pdf

[35] Rabbit Semiconductor, "RCM3750 RabbitCore User's Manual," September 2005 http://www.rabbitsemiconductor.com/documentation/docs/manuals/RCM3750/RC3750UM.pdf

[36] Rabbit Semiconductor, "Dynamic C TCP/IP User's Manual," February 2005 http://www.rabbitsemiconductor.com/documentation/docs/manuals/TCPIP/UsersManualV2/tcpV2.pdf

[37] TCP/IP Protocols frame format, Date Retrieved: October, 2006, http://securitywizardry.com/protpackets.htm

APPENDIX A - TCP/IP Protocol Frame Formats

Appendix A provides all important frame formats used in the TCP/IP implementation of an embedded web server.

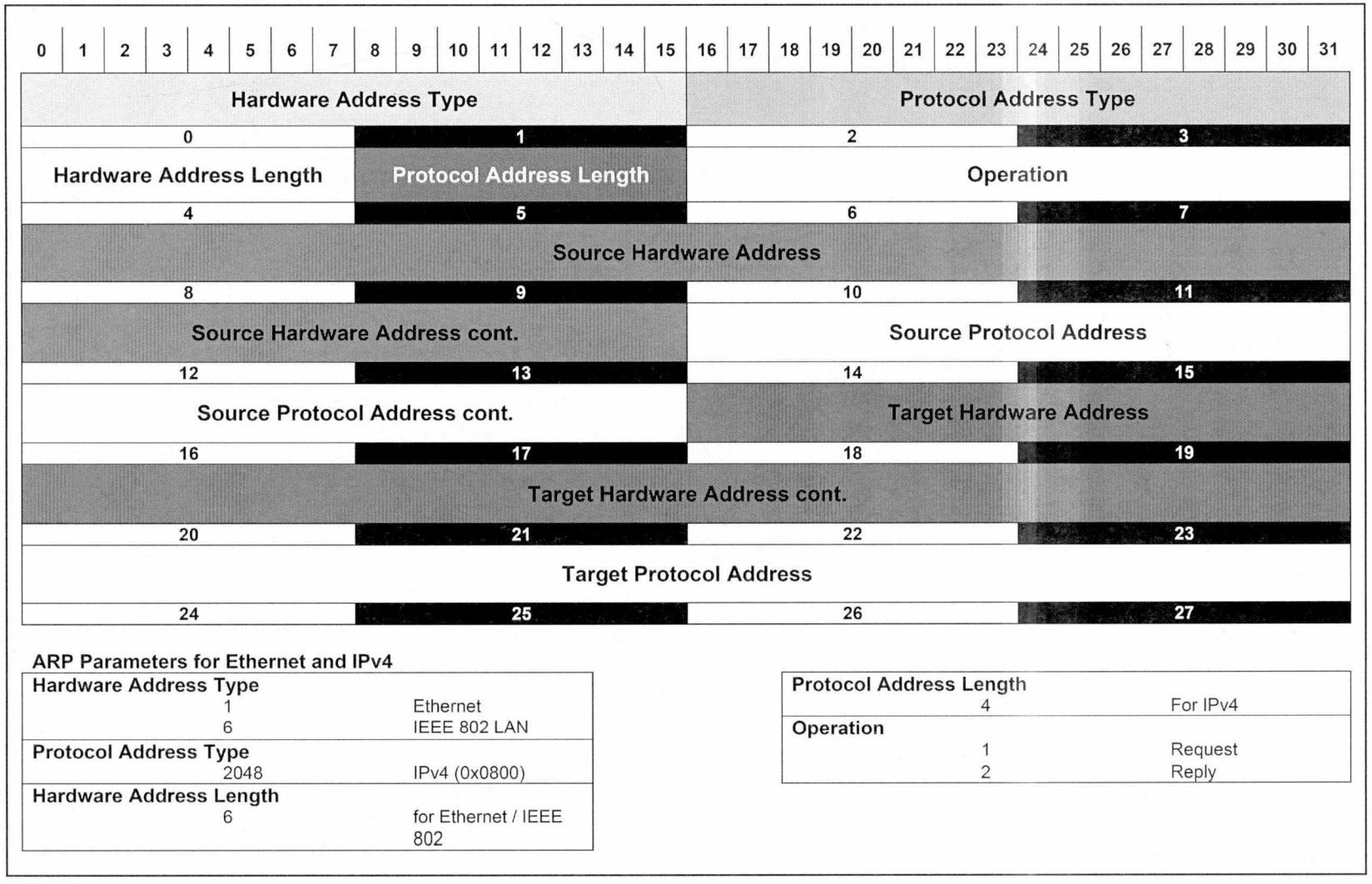
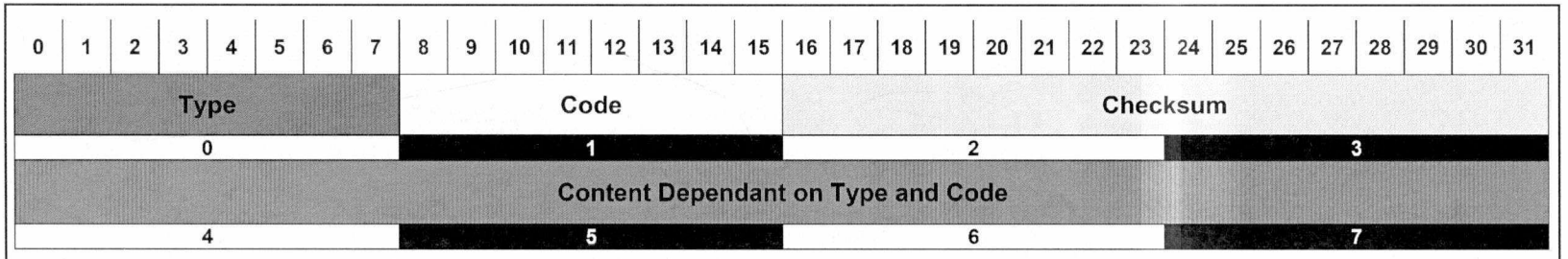


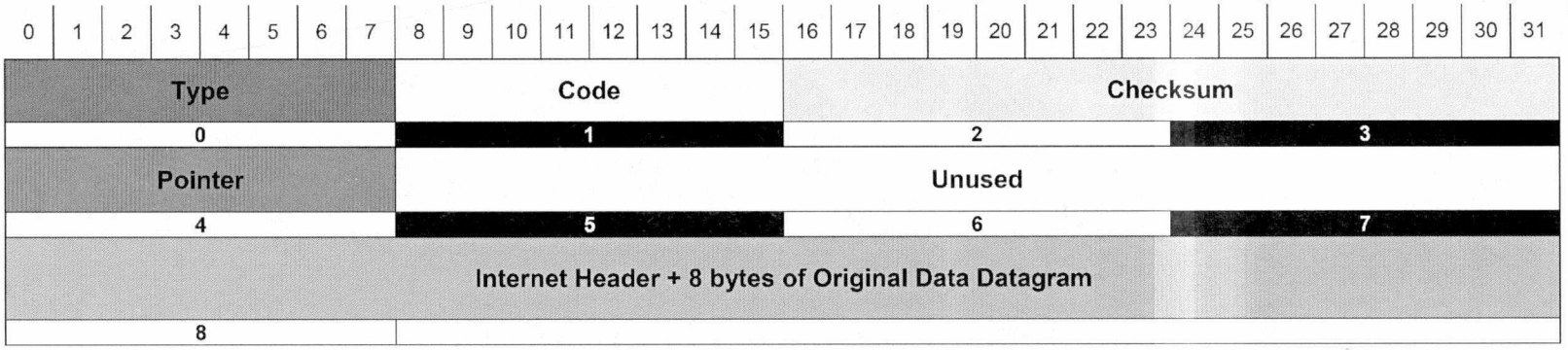
Figure A-1: Address Resolution Protocol message format [37]



Type	Code	Meaning
0	0	Echo Reply
3	0	Net Unreachable
	1	Host Unreachable
	2	Protocol Unreachable
	3	Port Unreachable
	4	Frag needed and DF set
	5	Source route failed
	6	Dest network unknown
	7	Dest host unknown
	8	Source host isolated
	9	Network admin prohibited
	10	Host admin prohibited
	11	Network unreachable for TOS
	12	Host unreachable for TOS
	13	Communication admin prohibited
4	0	Source Quench (Slow down/Shut up)

Type	Code	Meaning
5	0	Redirect datagram for the network
	1	Redirect datagram for the host
	2	Redirect datagram for the TOS & Network
	3	Redirect datagram for the TOS & Host
8	0	Echo
9	0	Router advertisement
10	0	Router selection
11	0	Time To Live exceeded in transit
	1	Fragment reassemble time exceeded
12	0	Pointer indicates the error (Parameter Problem)
	1	Missing a required option (Parameter Problem)
	2	Bad length (Parameter Problem)
13	0	Time Stamp
14	0	Time Stamp Reply
15	0	Information Request
16	0	Information Reply
17	0	Address Mask Request
18	0	Address Mask Reply
30	0	Traceroute (Tracert)

Figure A-2: Internet Control Message Protocol message format [37]



Type	Code	Meaning
0	0	Echo Reply
3	0	Net Unreachable
	1	Host Unreachable
	2	Protocol Unreachable
	3	Port Unreachable
	4	Frag needed and DF set
	5	Source route failed
	6	Dest network unknown
	7	Dest host unknown
	8	Source host isolated
	9	Network admin prohibited
	10	Host admin prohibited
	11	Network unreachable for TOS
	12	Host unreachable for TOS
	13	Communication admin prohibited
4	0	Source Quench (Slow down/Shut up)

Type	Code	Meaning
5	0	Redirect datagram for the network
	1	Redirect datagram for the host
	2	Redirect datagram for the TOS & Network
	3	Redirect datagram for the TOS & Host
8	0	Echo
9	0	Router advertisement
10	0	Router selection
11	0	Time To Live exceeded in transit
	1	Fragment reassemble time exceeded
12	0	Pointer indicates the error (Parameter Problem)
	1	Missing a required option (Parameter Problem)
	2	Bad length (Parameter Problem)
13	0	Time Stamp
14	0	Time Stamp Reply
15	0	Information Request
16	0	Informaiton Reply
17	0	Address Mask Request
18	0	Address Mask Reply
30	0	Traceroute (Tracert)

Figure A-3: Internet Control Message Protocol parameter message format [37]

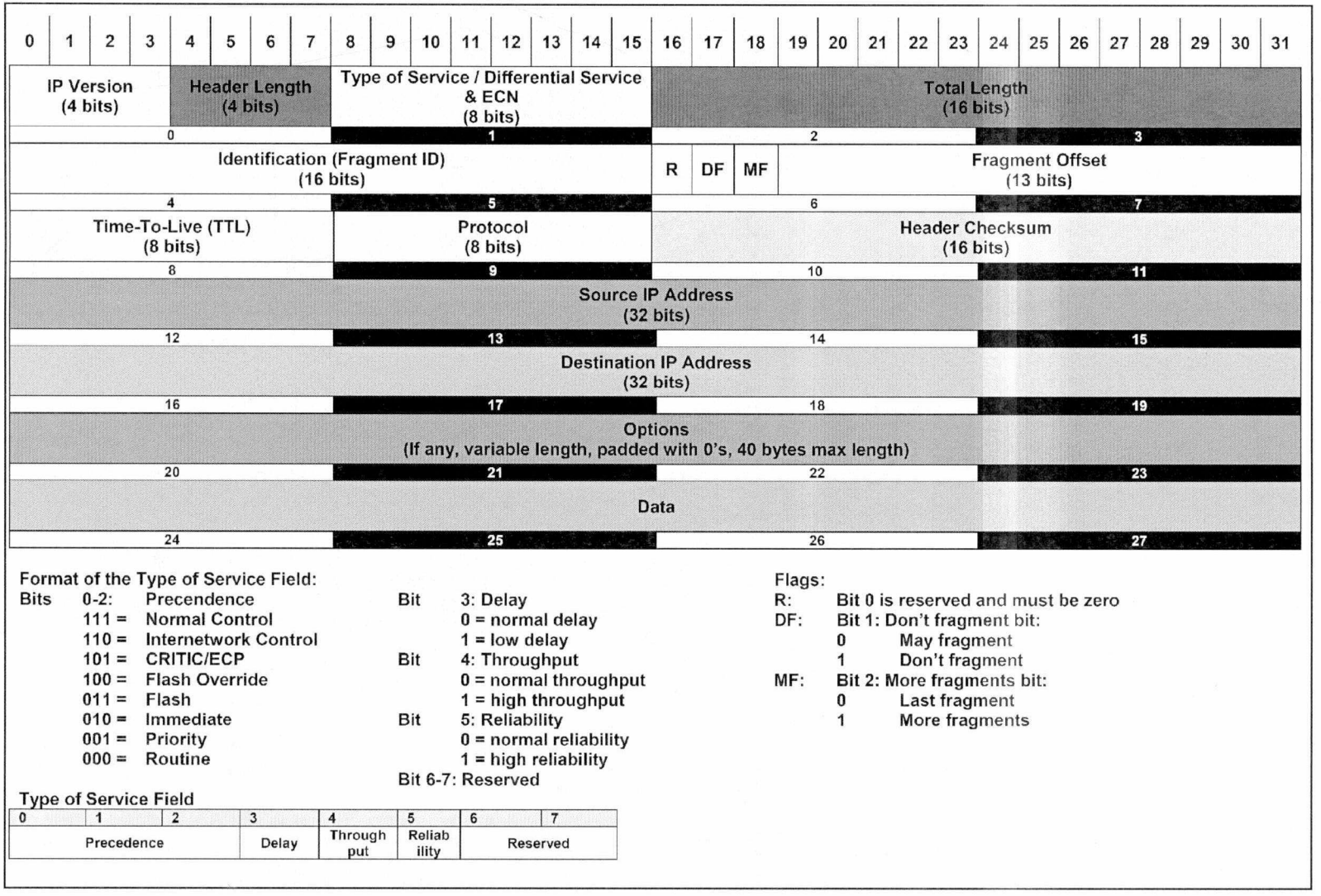


Figure A-4: Internet Protocol message format [37]

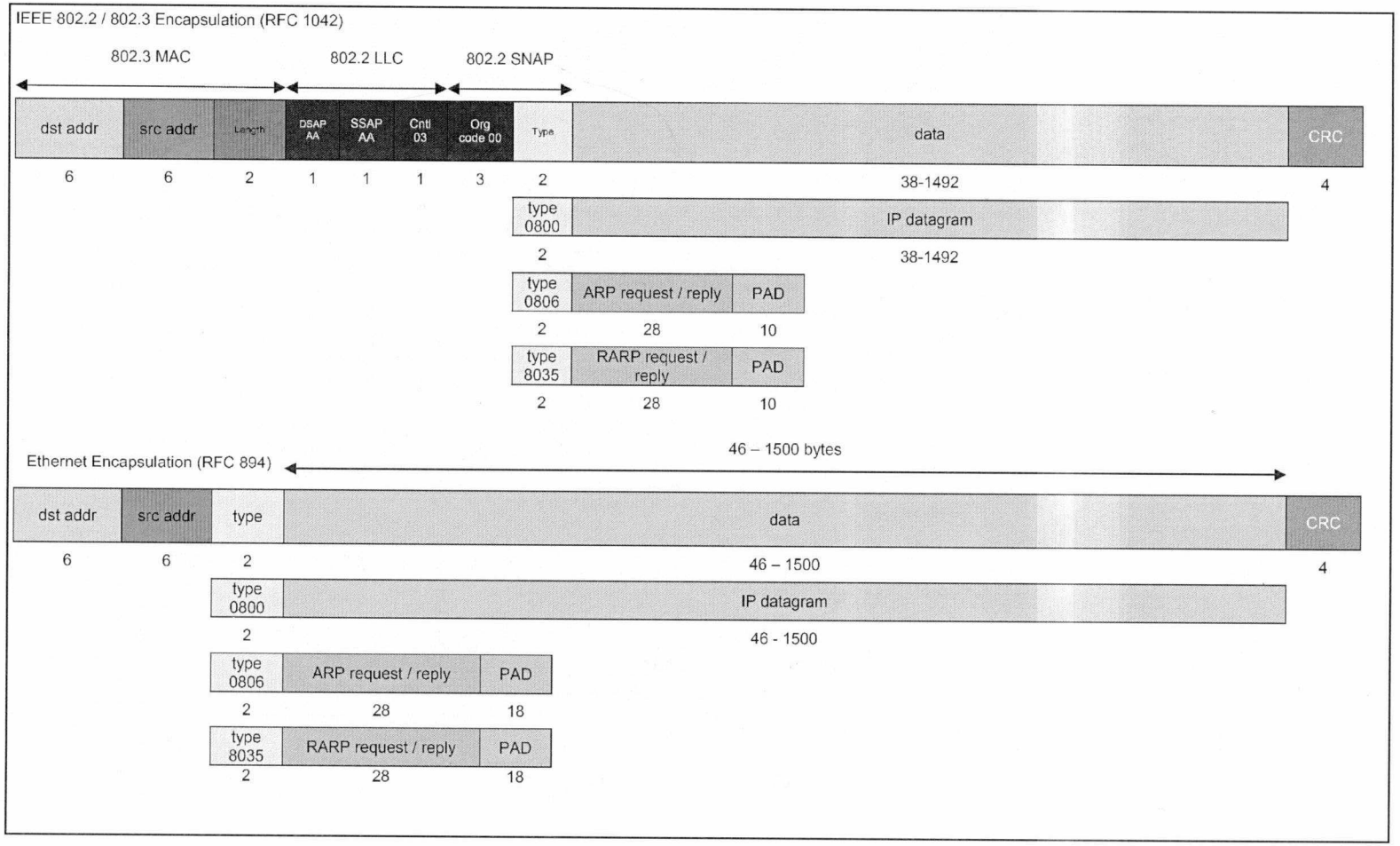


Figure A-5: Link layer header format [37]

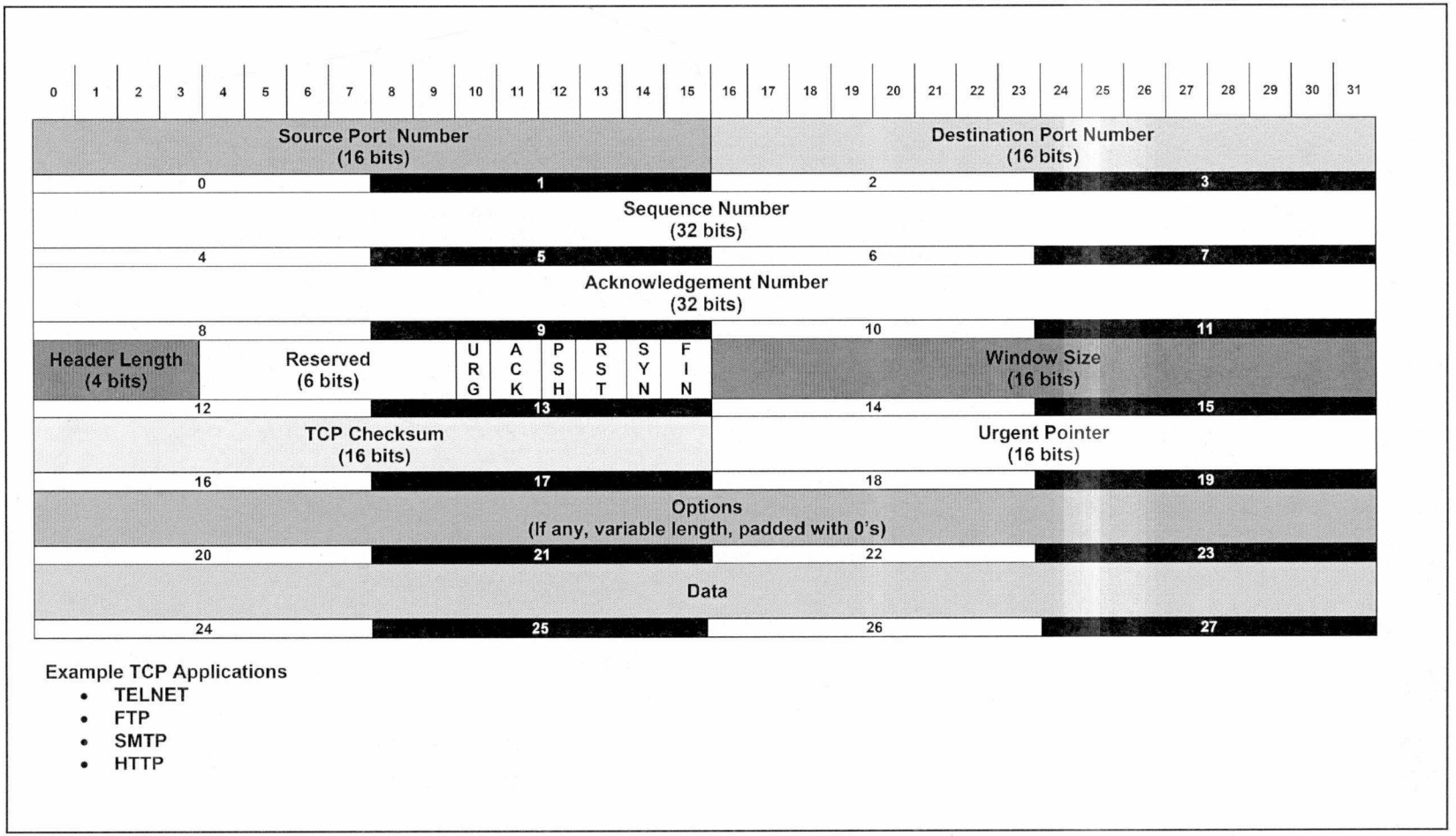
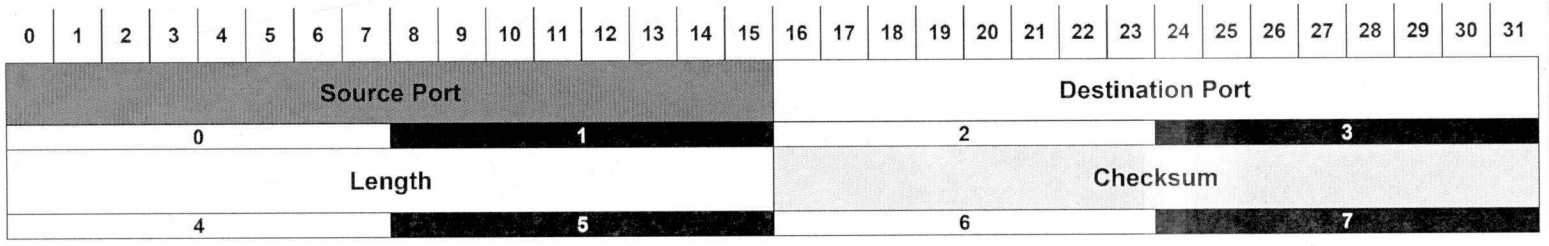


Figure A-6: Transmission Control Protocol message format [37]



Common UDP Well-Known Server Ports

Port	Description
7	Echo
19	Chargen
37	Time
53	Domain
67	Bootps (DHCP)
68	Bootpc (DHCP)
69	Tftp
137	Netbios-ns

Port	Description
138	Netbios-dgm
161	Snmp
162	Snmp-trap
500	Isakmp
514	Syslog
520	Rip
33434	Traceroute

Length
The number of bytes in the entire datagram, including the header; minimum value = 8

Checksum
Covers pseudo-header and entire UDP datagram

Figure A-7: User Datagram Protocol message format [37]

APPENDIX B - Embedded Web Server Snapshots

Appendix B provides the snapshots of the embedded web server operating modes and the supporting features to web enable a distributed wireless sensor network.

B.1 Embedded Web Server – Security Features

The embedded web server implements security features to limit the access to the web server resources and critical network controls of the wireless sensor network. The web server access controls are greatly extended by adding three different set of user groups. Setting the operating mode of the server is one of the critical controls and only an administrator or an authorized user is permitted to change the server operating mode. The operating mode of the web server is not set when the server is powered on or reset. Figure B-1 shows the snapshot of the server home page as soon as the server is powered on.

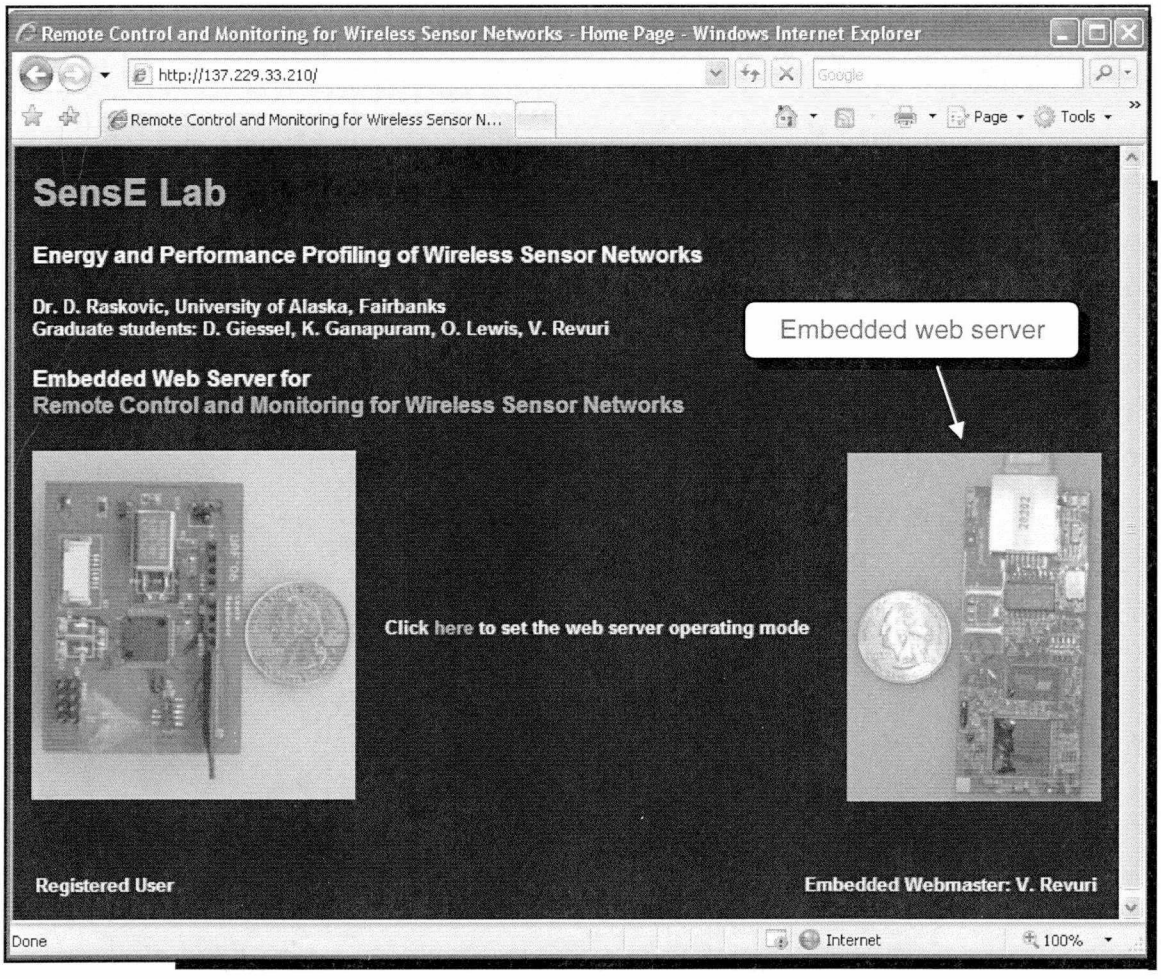


Figure B-1: Embedded web server home page

The administrator or a registered user can set the operating mode of the server by following the link to the web server controls configuration web page. The server uses basic authentication method to apply access permissions to the resources in the file system. The permissions and authorizations are set to the web pages to restrict its access to the clients. Figure B-2 shows the screenshot of browser requesting the administrator user name and password to access the web server's configuration page. The administrator can access all the critical controls of the web server and the wireless sensor network through the server's configuration page.

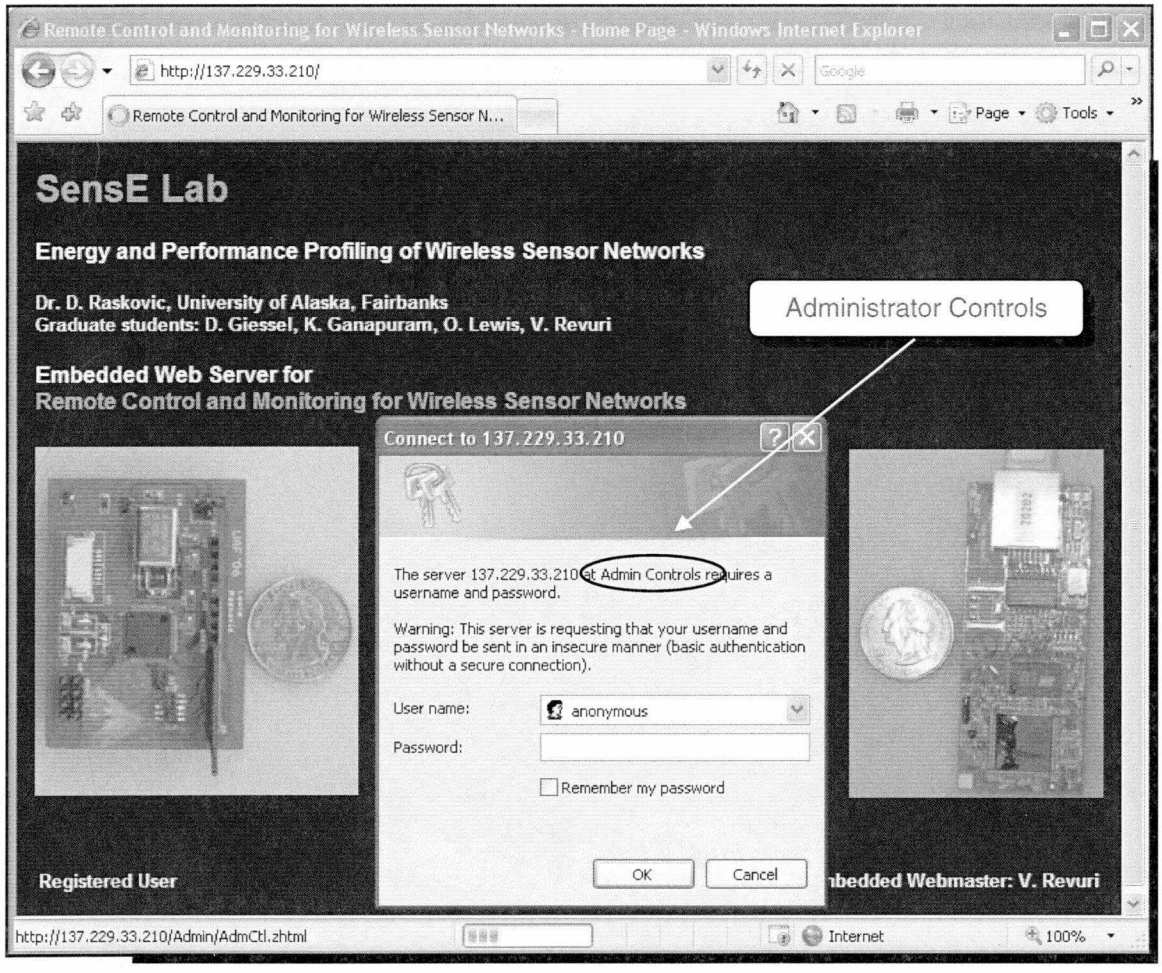


Figure B-2: Server administrator authentication request

Figure B-3 shows the screenshot of browser requesting the registered user name and password to access the web server's configuration page.

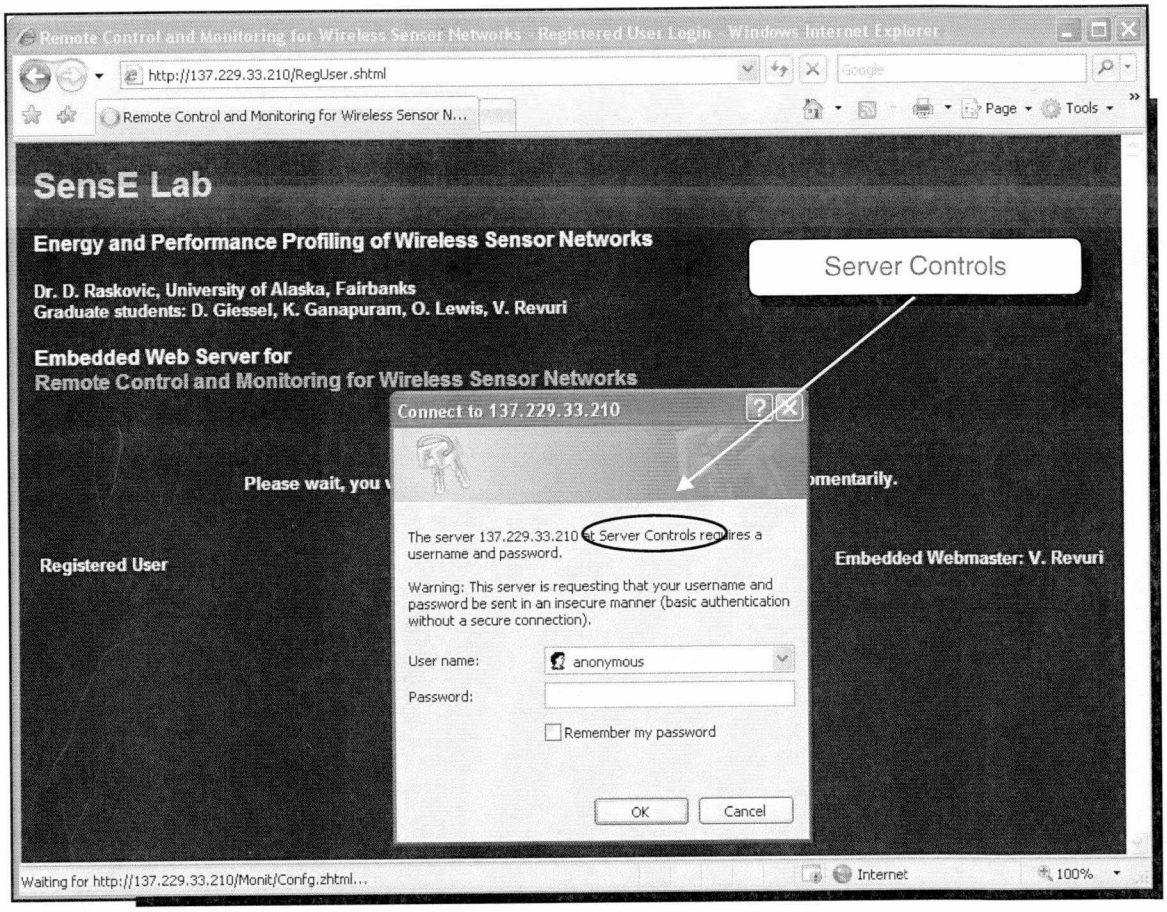


Figure B-3: Registered user authentication request

Figure B-4 shows all the web server controls to monitor, control and collect data from the wireless sensor network. The data monitoring controls include snapshot mode and monitoring mode. The data collecting controls include the log mode for collecting the data and session request for the logged data. Other critical functions on the server configuration page include web server controls and sensor network controls.

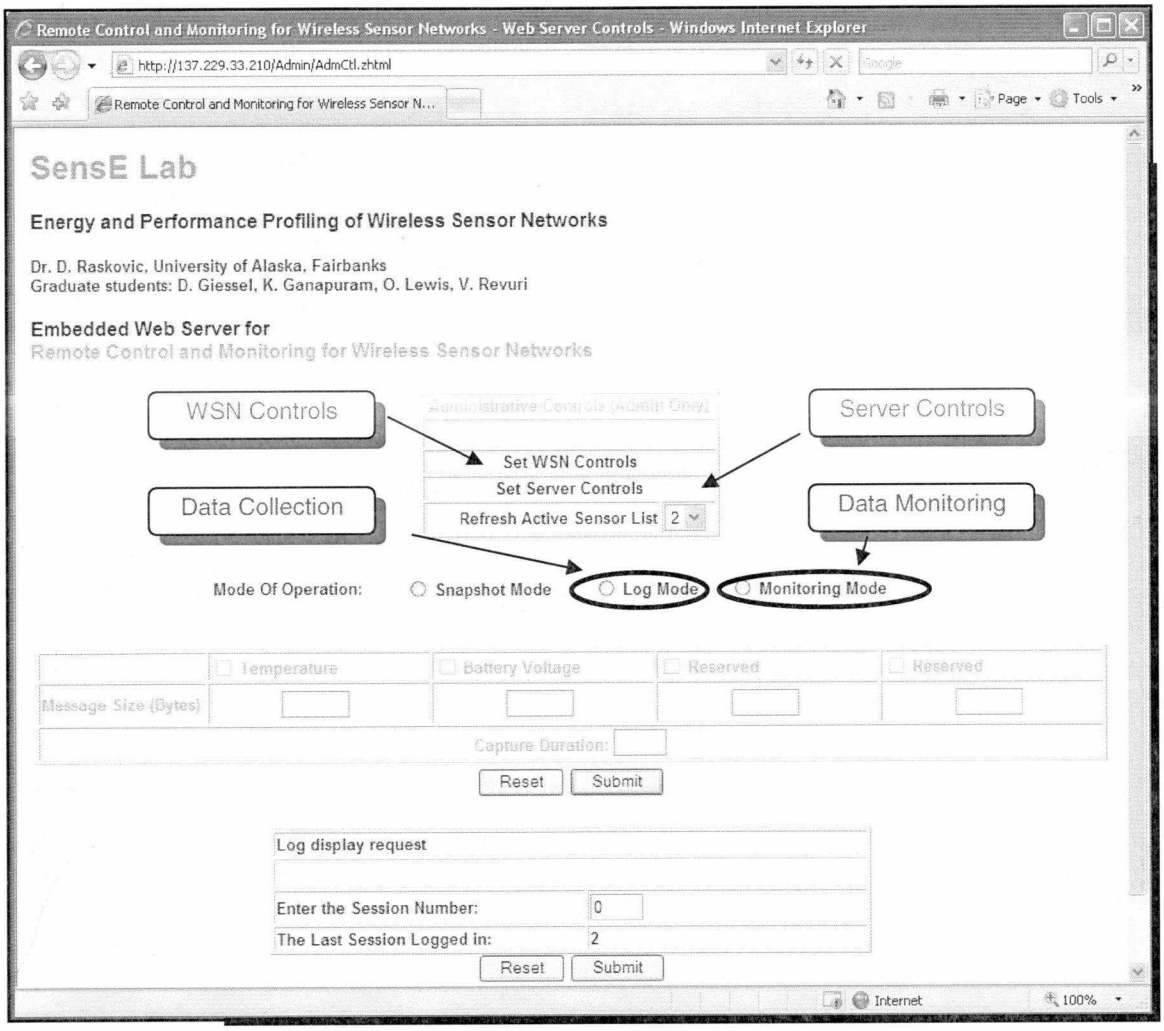


Figure B-4: Embedded web server configuration page

The administrator has an unrestricted access to the server and sensor network controls. The web server controls include resetting the operating mode of the server, clearing the log, configuring the email alerts, and setting the date of the server. Figure B-5 shows the snapshot of the wireless sensor network controls. The administrator can access and configure critical controls of the sensor network. The sensor network controls include radio configurations and network data configurations.

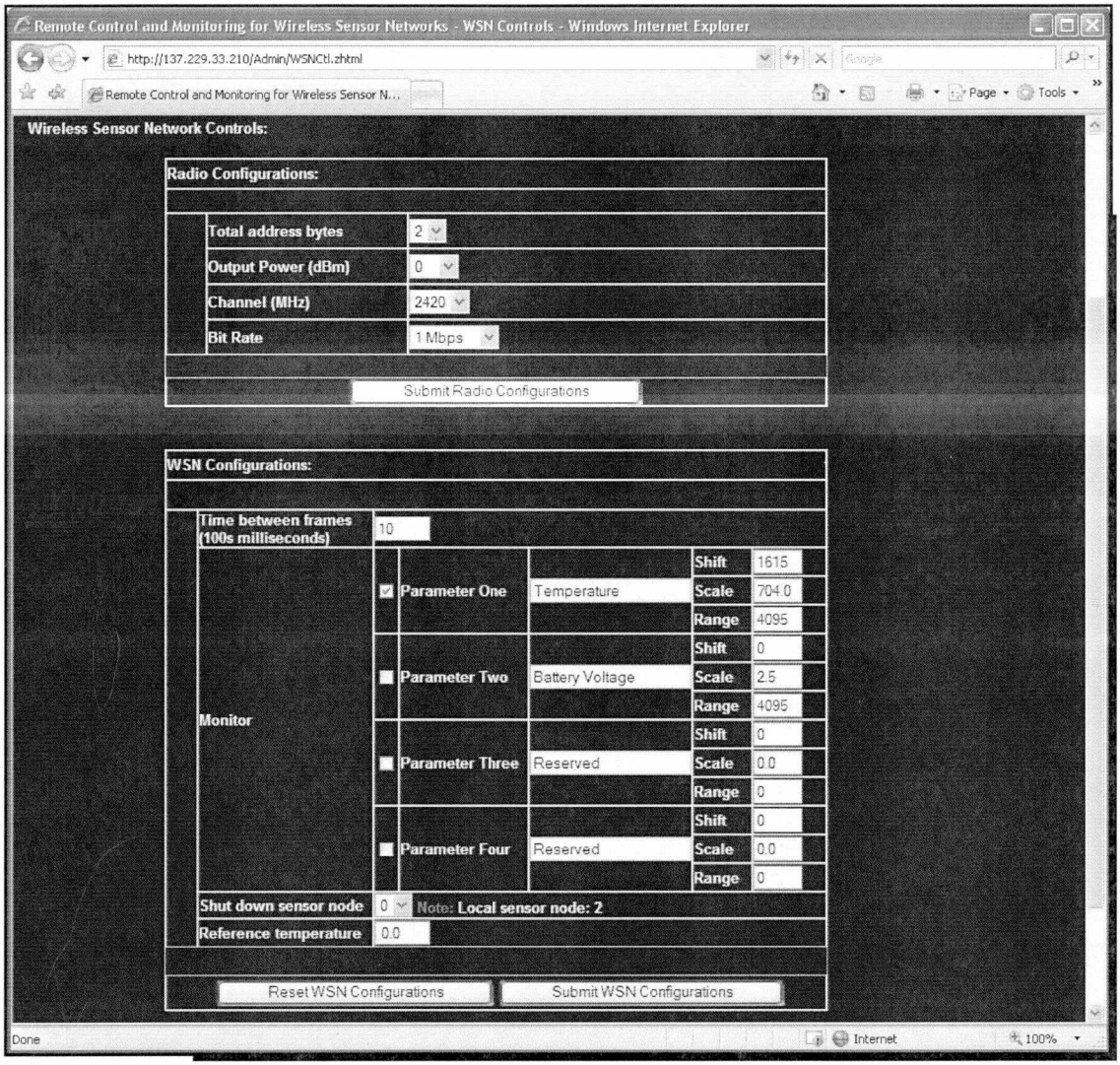


Figure B-5: Wireless Sensor Network controls

B.2 Embedded Web Server – Snapshot Mode

The snapshot mode is one of the data monitoring functionalities of the embedded web server and is used to get a glimpse of the network parameters. Figure B-6 shows the title page of the web server operating in the snapshot mode. The sensor nodes are identified by the sensor IDs. The web server periodically requests the list of sensor nodes present in the network and updates the list in the web page by dynamically inserting it into the page. The snapshot below shows the title web page when the server is operating in snapshot mode.

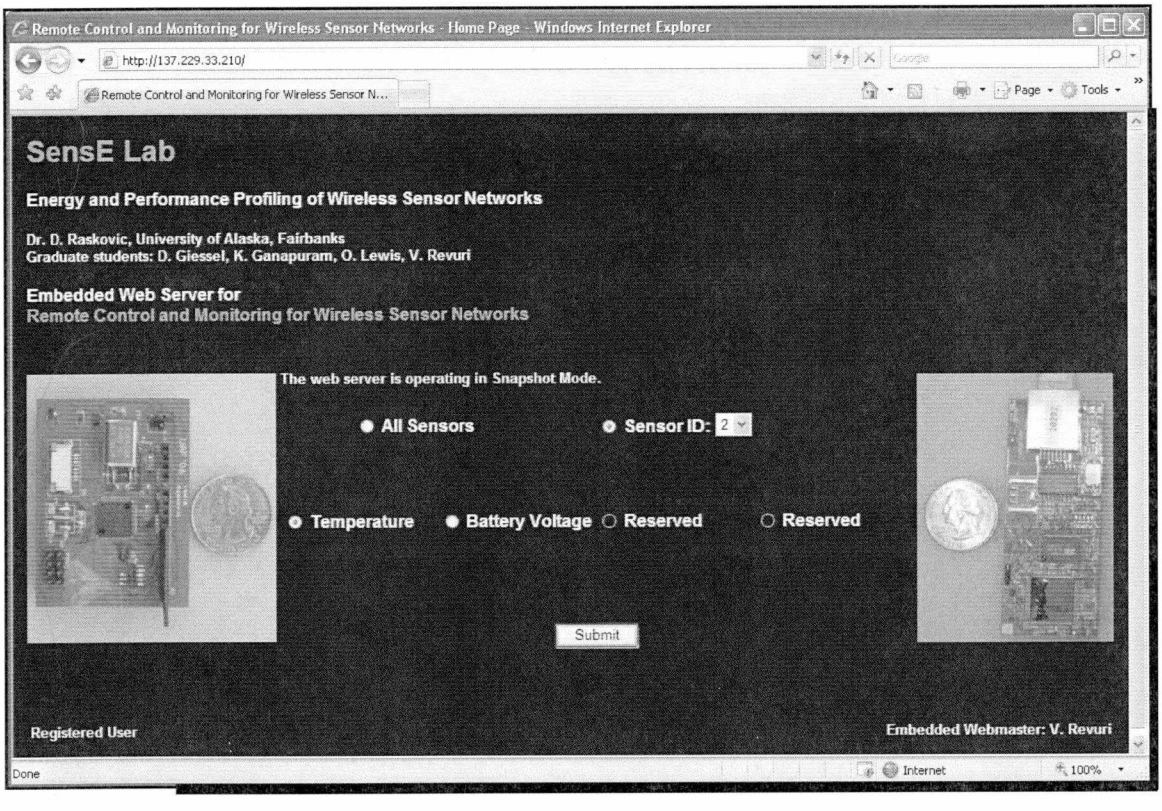


Figure B-6: Embedded web server operating in snapshot mode

In the snapshot mode, the server passively listens to the requests from the clients and when the network monitored data is requested, the server retrieves the data from the wireless sensor network and responds with a graphical representation of the network data. The snapshot mode has an option of either requesting parameter value of a single sensor

node or all active sensors in the network. Figure B-7 shows the response to the request for parameter one (temperature) of a sensor node 2.

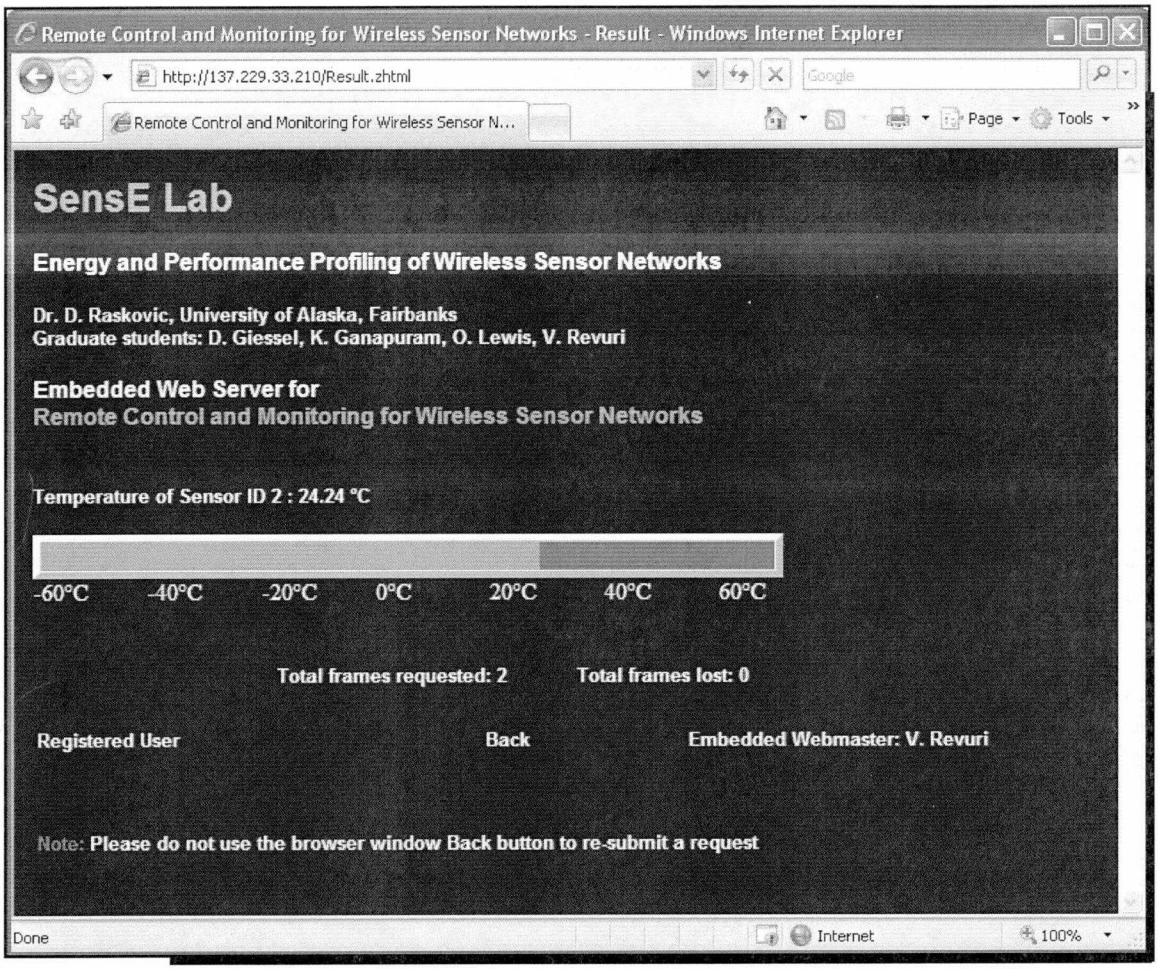


Figure B-7: Single sensor node parameter response

B.3 Embedded Web Server – Monitoring Mode

The server is operated in monitoring mode by selecting the mode of operation radio button in the server configuration to monitoring mode and the client should also specify the parameters to monitor. In the monitoring mode, the server requests the selected network data regularly at configured time intervals and stores the latest 20 parameter values of all the parameters. Figure B-8 shows the web server's monitoring mode radio button and parameter checkboxes needed to operate the server in monitoring mode. Figure B-9 shows the response to the parameter two (battery voltage) of sensor node 2 request.

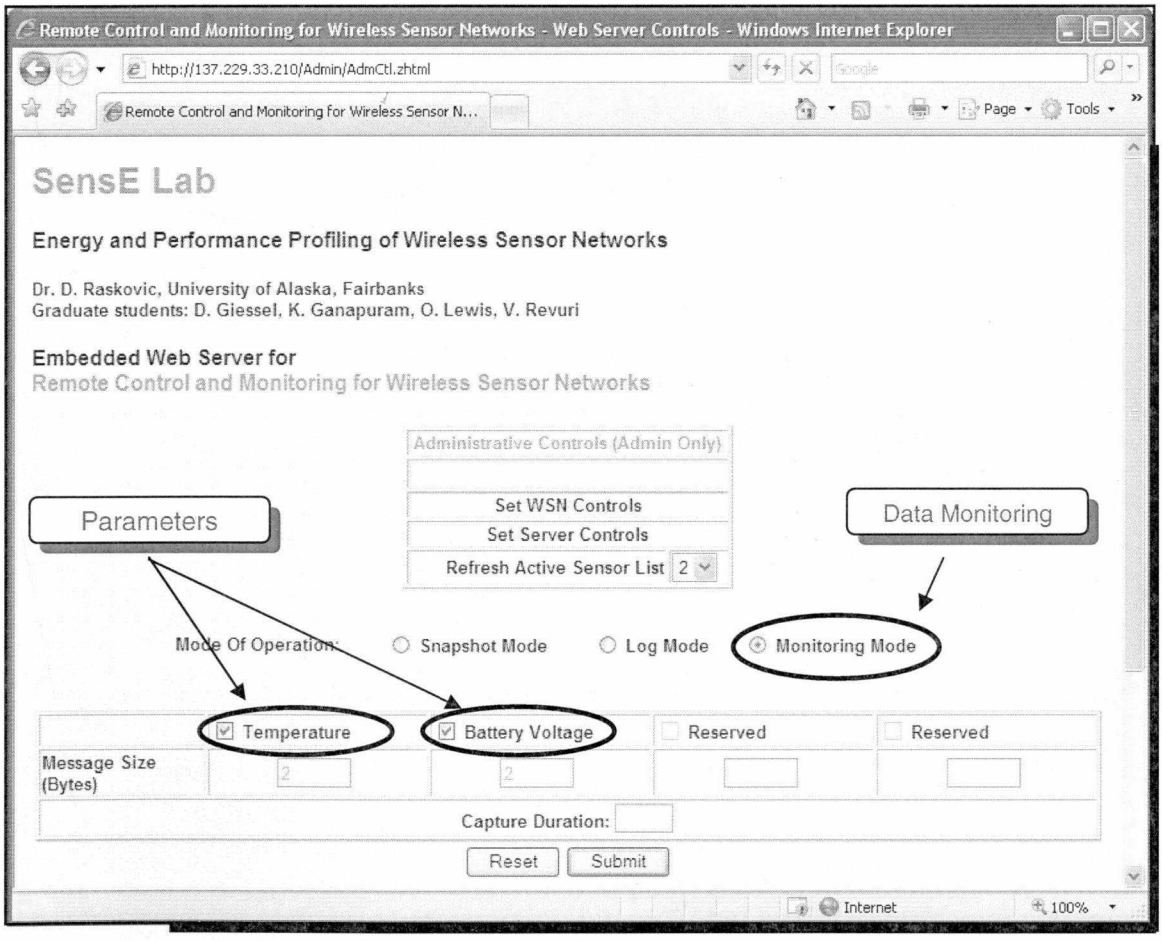


Figure B-8: Setting the controls to operate the web server in monitoring mode

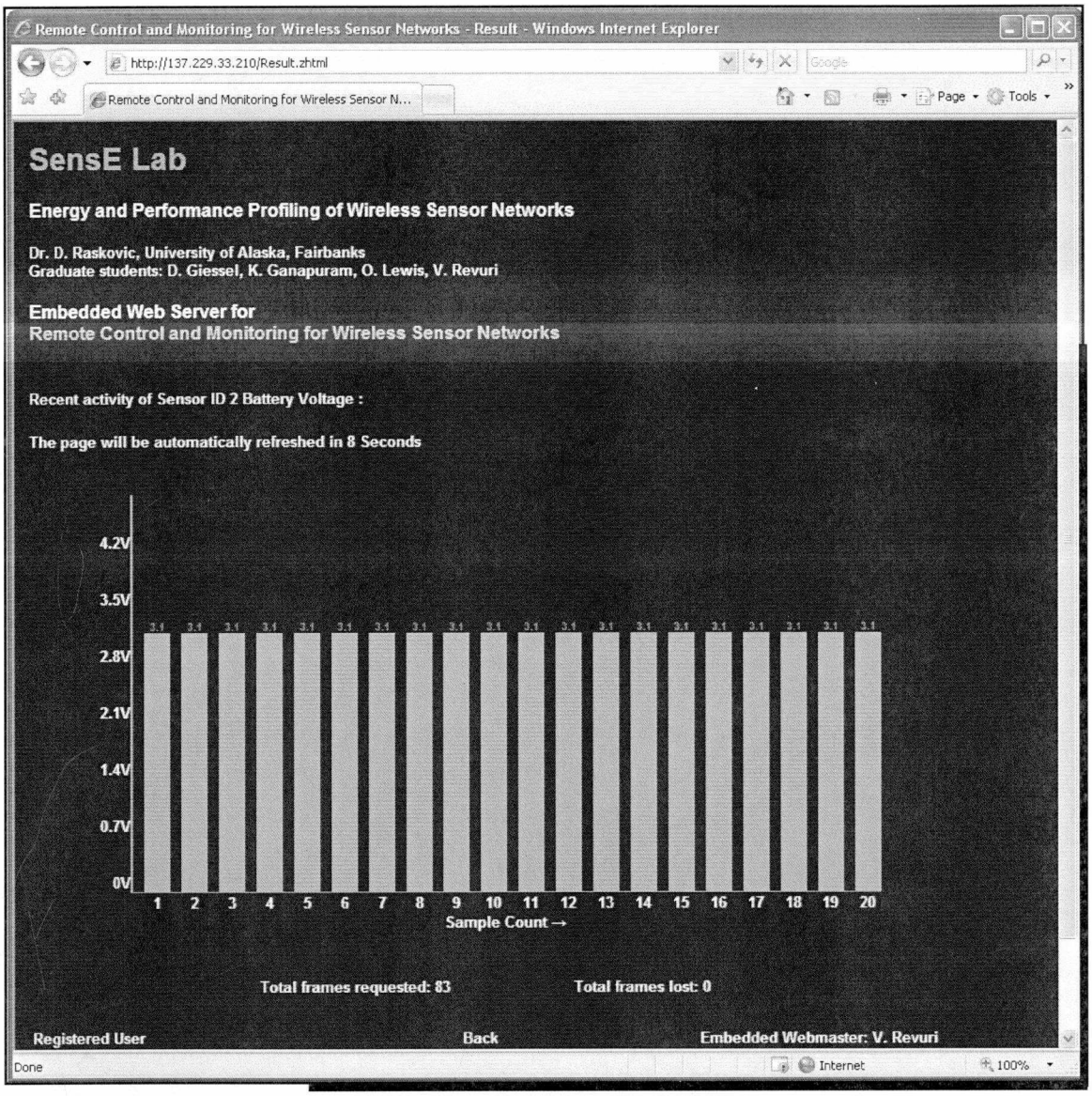


Figure B-9: The battery voltage display when the server is operated in monitoring mode

B.4 Embedded Web Server – Log Mode

The log mode fulfills the data collecting functionality of the embedded web server, this mode allows the server to collect the selected network monitored parameters and latter, the client can download the data and decode it using the application developed in MATLAB®. To operate the server in the log mode, the mode of operation radio button is selected to log mode, and the parameters to be collected are selected together with the duration of logging. Once the data is collected, it can be downloaded to the client's computer through the Internet or can be directly read from the server's serial port. Figure B-10 shows the logged data which is displayed in the textbox of the browser.

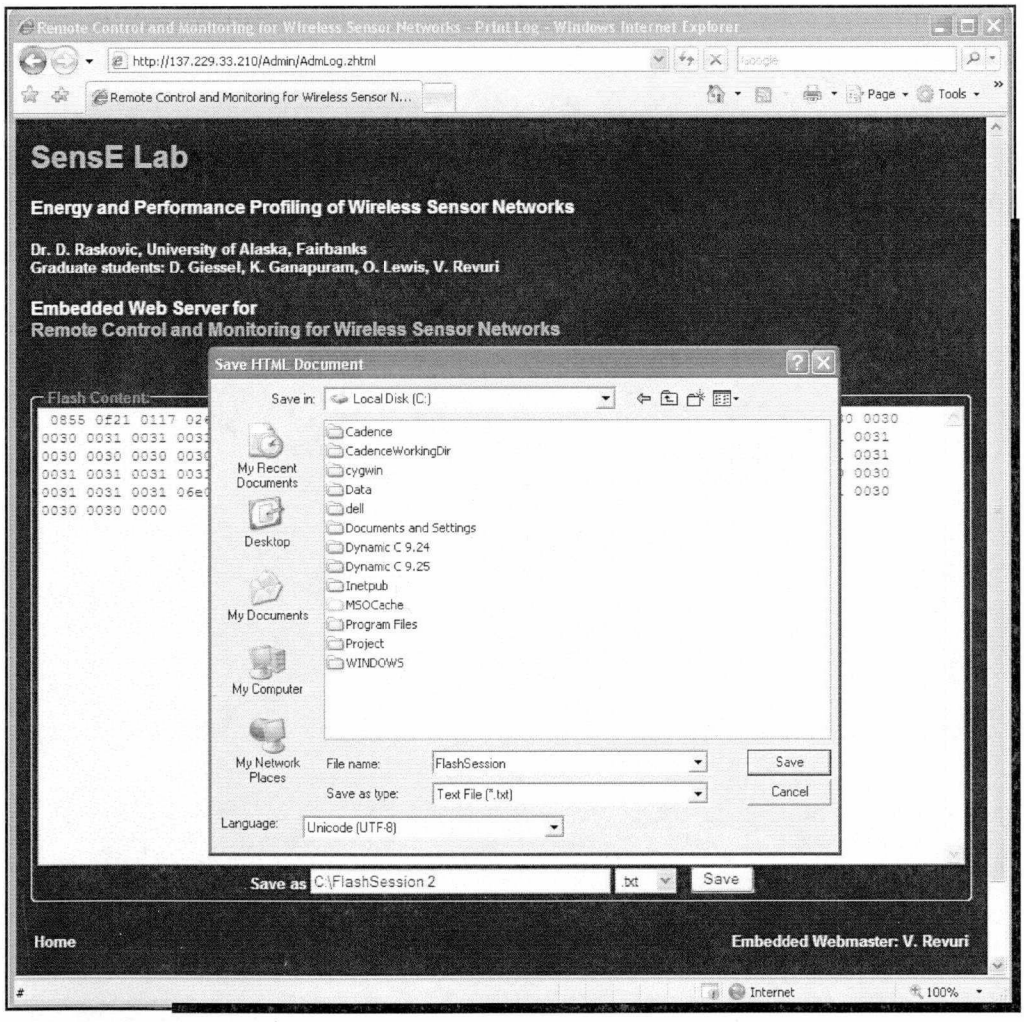


Figure B-10: The response to the log session request

The data can be saved to a text file by clicking on the save button on the browser and latter can be decoded using the application developed in MATLAB®. Figure B-11 shows the application interface developed in MATLAB® to decode the logged data.

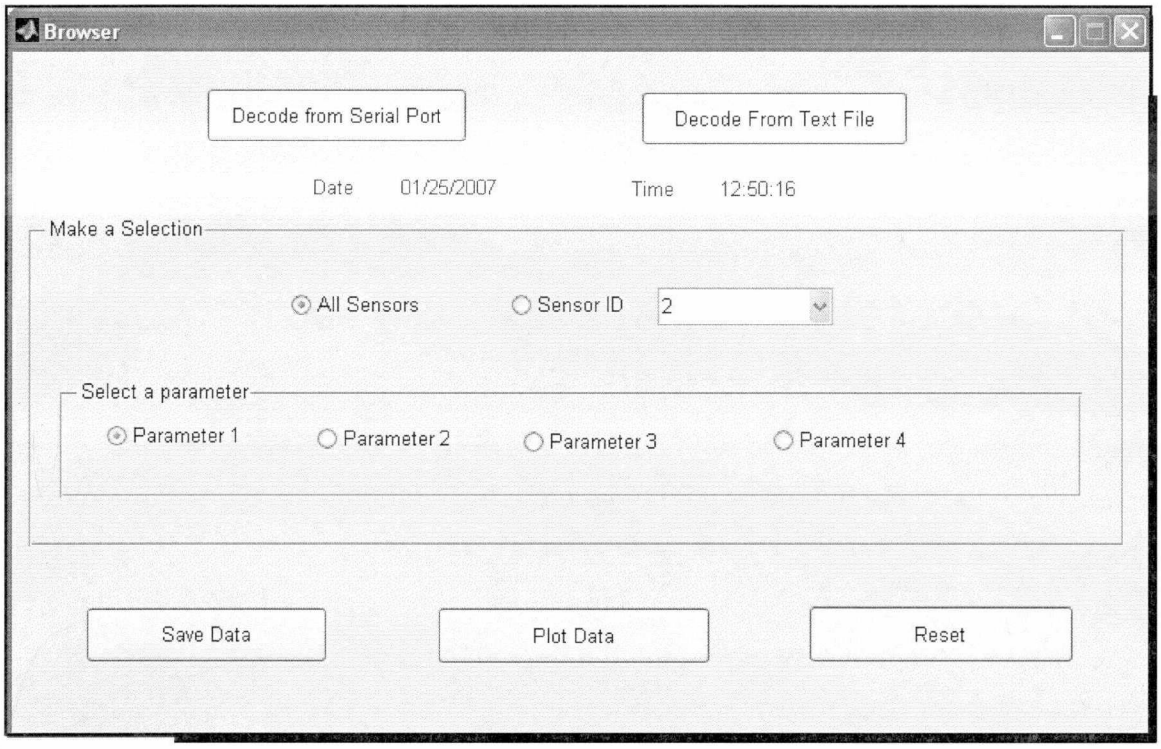


Figure B-11: MATLAB® application user-interface to decode the logged data

B.5 Embedded Web Server – Email Alerts

The web server keeps track of the wireless sensor network operations and network monitored data, and reports any critical issues developed during the network operations by sending an email to the authorized client. The server is capable of reporting the abnormal working of the wireless sensor network, web server's serial flash memory status, and the sensors nodes which reached the set threshold values. The email alert web page helps the user to set the conditions for sending an email. Figure B-12 shows the email alert page with the conditions to send a report the client.

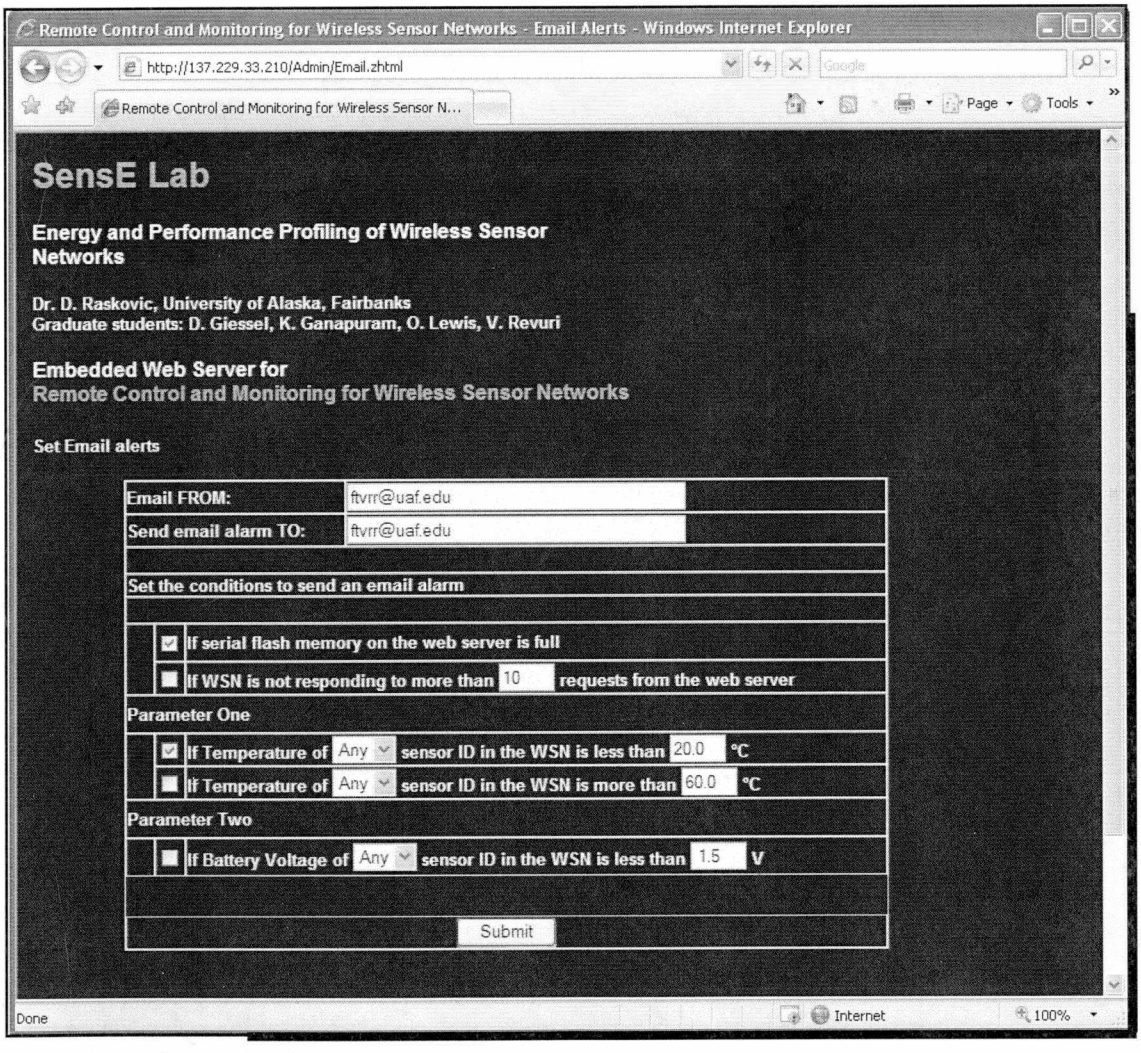


Figure B-12: Web server email alerts configuration page

Figure B-13 is an example of an email alert sent to the authorized client when the sensor network did not respond to the more than ten requests from the web server. The email includes the dynamic report of the email.

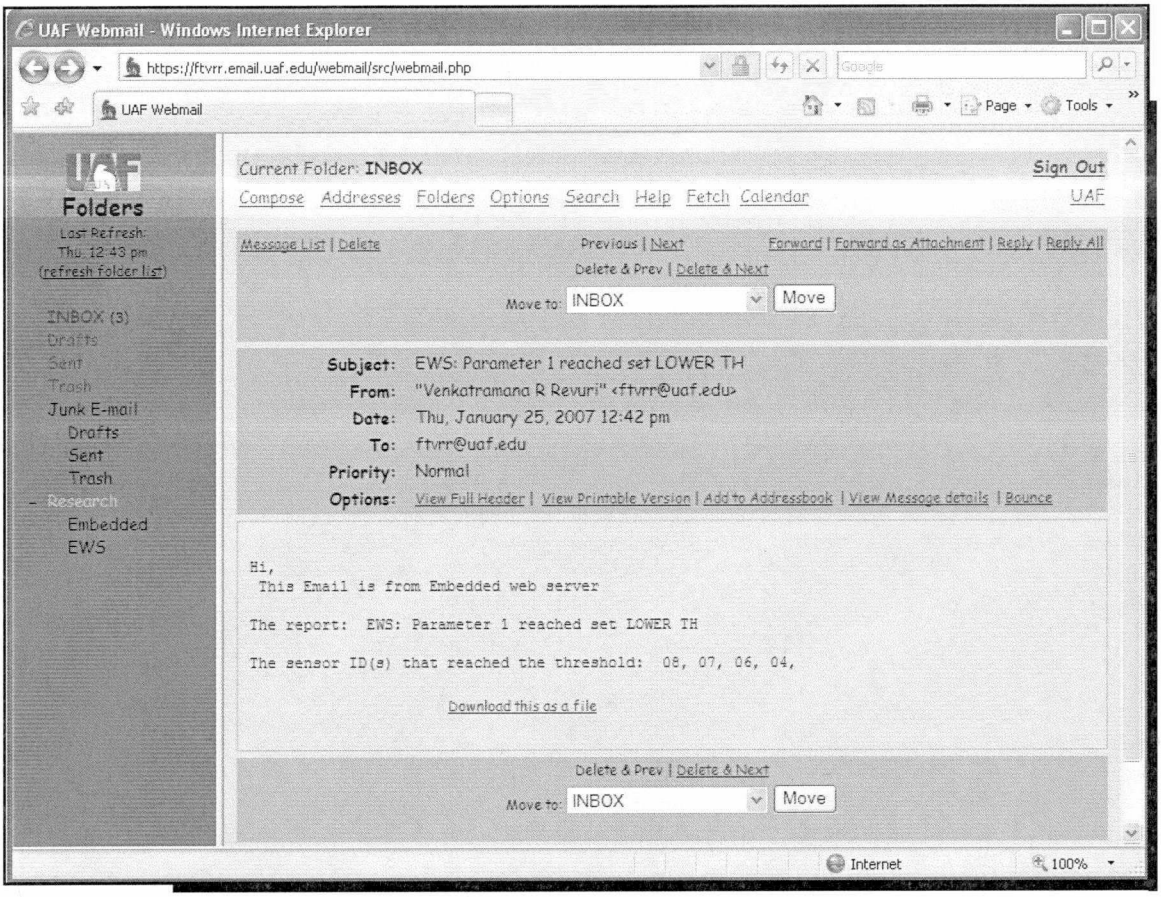


Figure B-13: An email alert sent to an authorized email client